Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage

Marie-Sarah Lacharité
marie-sarah.lacharite.2015@rhul.ac.uk

Brice Minaud
brice.minaud@rhul.ac.uk

Kenneth G. Paterson
kenny.paterson@rhul.ac.uk

Abstract

We analyse the security of database encryption schemes supporting range queries against persistent adversaries. The bulk of our work applies to a generic setting, where the adversary's view is limited to the set of records matched by each query (known as *access pattern* leakage). We also consider a more specific setting where certain *rank* information is also leaked. The latter is inherent to multiple recent encryption schemes supporting range queries, including Kerschbaum's FH-OPE scheme (CCS 2015), Lewi and Wu's order-revealing encryption scheme (CCS 2016), and the recently proposed Arx scheme of Poddar *et al.* (IACR eprint 2016/568, 2016/591). We provide three attacks.

First, we consider *full reconstruction*, which aims to recover the value of every record, fully negating encryption. We show that for dense datasets, full reconstruction is possible within an expected number of queries $N \log N + O(N)$, where N is the number of distinct plaintext values. This directly improves on a $O(N^2 \log N)$ bound in the same setting by Kellaris *et al.* (CCS 2016). We also provide very efficient, data-optimal algorithms that succeed with the minimum possible number of queries (in a strong, information theoretical sense), and prove a matching data lower bound for the number of queries required.

Second, we present an *approximate reconstruction* attack recovering all plaintext values in a dense dataset within a constant ratio of error (such as a 5% error), requiring the access pattern leakage of only $O(N)$ queries. We also prove a matching lower bound.

Third, we devise an attack in the common setting where the adversary has access to an *auxiliary distribution* for the target dataset. This third attack proves highly effective on age data from real-world medical data sets. In our experiments, observing only 25 queries was sufficient to reconstruct a majority of records to within 5 years.

In combination, our attacks show that current approaches to enabling range queries offer little security when the threat model goes beyond snapshot attacks to include a persistent server-side adversary.

1 Introduction

Various kinds of property-preserving encryption (PPE) schemes have started to see wide deployment, in particular in the area of data storage outsourcing. There, a client encrypts a set of records or documents using a PPE scheme, sends it to the server, and is later able to query the server and retrieve the matching records or documents. By exploiting the special properties of the encryption scheme, the server can index the data just as it would unencrypted data, allowing the server to support efficient search. For example, deterministic encryption allows matching queries to be made, while Order-Preserving/Revealing Encryption (OPE/ORE) allow range queries to be efficiently supported.

At the same time, our understanding of the security that such schemes offer against various kinds of adversary is still developing. This has led to serious attacks being found against some of the early schemes [CGPR15, NKW15, GMN$^+$16, PW16, ZKP16, DDC16, GSB$^+$17] – a good summary of this line of research is available in [FVY+17]. A second generation of schemes, which typically use custom indexes rather than legacy indexes, promise to do better, in the sense of provably leaking less information about encrypted data. Perhaps inevitably, the second generation of schemes has been followed by another wave of attacks. Kellaris, Kollios, Nissim, and O'Neill introduced generic reconstruction attacks applicable to any scheme whose range queries leak access pattern or communication volume, for a uniform range query distribution [KKNO16]. Grubbs, Sekniqi, Bindschaedler, Naveed, and Ristenpart presented a snapshot attack on non-deterministic, frequency-hiding OPE schemes when auxiliary information about the plaintext distribution is available [GSB+17]. We continue this line of research into generic attacks, which apply even to second-generation encryption schemes, focussing on those schemes that support range queries.

1.1 Setting and Notation

Our attacks share the same general setting, which we now describe, together with the relevant notation. We let $[a, b]$ denote the set of integers $\{a, \dots, b\}$ (by convention $[a, b] = \varnothing$ whenever $a > b$).

Records, identifiers, and values. We consider a collection of R records (or documents) in a database, each with a unique identifier. The set of record identifiers is assumed, without loss of generality, to be $\mathcal{R} = [1, R]$. Each record $r \in \mathcal{R}$ contains a field with a single value $\mathsf{val}(r)$ from some ordered set of values $\mathcal{X}$ of size N, on which range queries are performed. We assume, again without loss of generality, that $\mathcal{X} = [1, N]$, with "<" being the usual ordering on the integers. We let $\mathcal{S}_x \subseteq [1, R]$ denote the set of identifiers of all records that contain value x. Each value can of course appear in more than one record.

In our attacks, it is assumed that the set of all record identifiers is known.(This assumption is discussed in Section 4.2 – in any case, with uniform queries, the set of all identifiers is recovered within $O(N)$ queries.)

Range queries. A range query $[x, y]$ is defined by its two end points $x \leq y$ in $\mathcal{X}$. In our analyses, except where indicated otherwise, range queries are modelled as uniformly distributed non-empty intervals in $[1, N]$, denoted by the distribution $\mathcal{U}$. The uniformity assumption is briefly discussed in the next section. A range query returns, at the very least, a set of matching record identifiers $\mathcal{M} := \{r \in \mathcal{R} : \mathsf{val}(r) \in [x, y]\}$. We also write $\mathcal{M} = \mathcal{S}_{[x,y]} := \cup_{x \leq t \leq y} \mathcal{S}_t$.

Adversarial model. The adversarial setting we consider throughout is that of a persistent, passive adversary, able to observe all communication between client and server. In the context of database encryption schemes, the word *persistent* is used by contrast with a *snapshot* adversary, who would only have access to a single snapshot of the server's memory at a particular time. The goal of the adversary is to reconstruct some information about the (plaintext) content of the encrypted database hosted on the server, *i.e.* infer information about the client's data.

Such a model captures two distinct threats. The first is that of an adversary having compromised the server for a sufficient length of time, or a man-in-the-middle attacker intercepting communications between client and server. The second, and more immediate one, is the scenario where the server itself is an honest-but-curious adversary: indeed, the server is obviously able to observe all communication between the client and itself. From this perspective, the security model we consider directly expresses the privacy of the user's data with respect to the server. In particular we emphasise that all our attacks could be mounted by the server itself (and ultimately show that the schemes under consideration offer little privacy for the user's data with the respect to the host server).

Since some database encryption scheme is used, the adversary is in general not able to observe queried ranges or other plaintext values. Instead, the view of the adversary is limited to some scheme-dependent *leakage* induced by each range query. In this article, two models are considered for this leakage, which we now present.

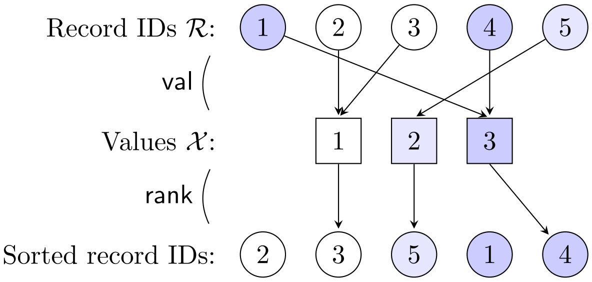


Figure 1: Example of mappings val and rank.

Access pattern leakage. As explained above, it is assumed that an adversary is able to observe some information leaked by range queries. If the leakage is limited to the set of matching record identifiers $\mathcal{M}$ as defined above, following [KKNO16], we call this leakage *access pattern* leakage. As discussed in [KKNO16], access pattern leakage can stem from the actual access pattern of the server, so long as memory accesses reveal which records are read. In this context the ID of a record is its memory address. This is a rather generic setting: to the best of our knowledge, all known efficient schemes supporting range queries leak the access pattern.

Rank information leakage. In addition, some of our attacks exploit *rank information* leakage. Although an attacker does not see the exact values of the end points x and y, it may learn some information regarding their rank. The *rank* of a value $z \in \mathcal{X}$ is the number of records having a value less than or equal to z:

$$\mathsf{rank}(z) := |\mathcal{S}_{[1,z]}|.$$

The rank of a value z can be interpreted as the highest position or index a record in $\mathcal{S}_z$ can have in a list of records sorted by value. In such a list, the position of any record r having value $z = \mathsf{val}(r)$ must fall between $\mathsf{rank}(z-1)+1$ and $\mathsf{rank}(z)$, where by convention $\mathsf{rank}(0) := 0$. The exact format of rank leakage can vary, and we discuss how it is leaked for some particular schemes in Section 1.3.

A simple illustration is provided in Figure 1. Here we take $R = 5$, $N = 3$, with $\mathsf{val}(1) = \mathsf{val}(4) = 3$, $\mathsf{val}(2) = \mathsf{val}(3) = 1$, and $\mathsf{val}(5) = 2$. In that case, as pictured, we have $\mathsf{rank}(1) = 2$, $\mathsf{rank}(2) = 3$, and $\mathsf{rank}(3) = 5$.

For a query on range $[x, y]$ (unknown to the attacker) matching records $\mathcal{M}$, the rank leakage is the values $a := \mathsf{rank}(x-1)$ and $b := \mathsf{rank}(y)$. The values $a+1$ and b can be interpreted as the lowest and highest positions (inclusive) of records in $\mathcal{M}$, within a list of records sorted by value. Note that the number of records returned by the query on range $[x, y]$ satisfies:

$$\begin{aligned} |\mathcal{M}| &= \textstyle\sum_{x \le t \le y} |\mathcal{S}_t| \\ &= \mathsf{rank}(y) - \mathsf{rank}(x-1) \\ &= b - a. \end{aligned}$$

Whenever we consider intervals of *values* in $[1, N]$, we shall use the letters $[x, y]$; whenever we consider intervals of *ranks* (generally the image of some values in $[1, N]$ via rank), we shall use the letters $[a, b]$.

Overall, if leakage is limited to the access pattern, then a query on range $[x, y]$ leaks $\mathcal{M} = \mathcal{S}_{[x,y]}$. If, in addition, rank information leakage is available, then the leakage of a query on range $[x, y]$ is $(a, b, \mathcal{M})$ where $a = \mathsf{rank}(x-1)$, $b = \mathsf{rank}(y)$, and $\mathcal{M} = \mathcal{S}_{[x,y]}$. When a set of queries is considered as an input to an adversary observing their leakage, we will often assimilate the set of queries with their leakage. In this context we write the input of the adversary as a set of triplets $\mathcal{Q} = \{(a_k, b_k, \mathcal{M}_k)\}$ if rank information is available, or simply $\mathcal{Q} = \{\mathcal{M}_k\}$ if only the access pattern is available.

	Req'd leakage	Assumptions	Number of queries	
	Rank	Density	Sufficient	Necessary
Full reconstruction attack				
[KKNO16]	N	N	$O(N^4 \log N)$	$\Omega(N^4)$
[KKNO16]	N	Y	$O(N^2 \log N)$	-
Sec. 2.3 (Algo. 1)	Y	Y	$N(\log N + 2)$	$N \log(N)/2 - O(N)$
Sec. 2.4 (Algo. 2)	N	Y	$N(\log N + 3)$	$N \log(N)/2 - O(N)$
Approximate reconstruction attack (within ϵN)				
Sec. 3.2 (Algo. 3)	N	Y	$\frac{5}{4} N \log(1/\epsilon) + O(N)$	$N \log(1/\epsilon)/2 - O(N)$

Table 1: Comparison of full and approximate reconstruction attacks exploiting access pattern leakage on encryption schemes allowing range queries, for a set of N distinct plaintext values, with uniformly distributed queries.

Density of the database. We say that the database is *dense* iff every value $x \in [1, N]$ occurs in at least one record. We will generally assume that the database is dense, except in Section 4. Whenever this assumption is necessary, formal statements will always mention it explicitly.

Miscellaneous mathematical notation. The notation log will always refer to the natural logarithm, rather than logarithm in base 2. We write H_n for the n-th harmonic number $H_n = \sum_{k=1}^{n} \frac{1}{k}$. Whenever f is a mapping $A \to B$, if $S \subseteq A$, $f(A)$ denotes the image set $\{f(a) : a \in A\}$.

1.2 Our Contributions

In this paper, we analyse the security of encryption schemes supporting range queries against persistent adversaries. The bulk of our work applies to the generic setting where the view of the adversary is limited to the set of matched records per query (access pattern leakage). We additionally consider the special case where the leakage contains rank information. It is natural to consider such leakage in schemes supporting range queries, and our attacks directly apply to, among others, Lewi and Wu's ORE scheme, Arx-Range, and Kerschbaum's frequency-hiding OPE scheme; more applications are discussed in Section 1.3.

In total we present three different, but related, attacks: full reconstruction, approximate reconstruction, and reconstruction using auxiliary information. An overview of the characteristics of our first two attacks, juxtaposed with previous results, is given in Table 1. Note that without rank leakage, full reconstruction is only up to reflection. In Table 1, the exact upper bounds on the expected number of queries for the attacks in Sections 2.3 and 2.4 require $N \geq 27$ and $N \geq 26$ respectively.

Full reconstruction attack. We show that access pattern leakage is sufficient to allow *full reconstruction*, *i.e.* recovering the value of *all* records with complete accuracy (up to reflection), within an expected number of only $N \log N + O(N)$ queries, where N is the number of distinct values. This directly improves on a $O(N^2 \log N)$ bound by Kellaris *et al.* in the same setting [KKNO16]. Furthermore this can be achieved both *data-optimally*, in the sense that as soon as the available leakage is sufficient for full reconstruction in an information theoretic sense, our algorithms do succeed; and efficiently. We note that the hidden constants in our data requirements are quite small; a bound for the expected number of queries is $N(\log(N) + 3)$ for $N \geq 26$.

In the setting where rank leakage is also available, we give an algorithm for full reconstruction, with a slightly better bound of $N(\log(N) + 2)$ queries for $N \geq 27$. Moreover the algorithm is still data-optimal in the previous sense, and especially efficient (incurring very little overhead over reading the query leakage). Finally, we also prove a lower bound of $N \log(N)/2 - O(N)$ on the expected number queries required for full reconstruction from access pattern leakage, for any algorithm, with or without rank information.

As explained in the previous section, our attacks are in the fully passive honest-but-curious setting, wherein a server stores the encrypted database, and processes a set of queries from a client, but has no additional powers otherwise. In particular, the server cannot make chosen queries of its own (if it could, then there are trivial reconstruction attacks requiring only N queries); and no knowledge about the distribution of values is assumed. We require only two mild extra assumptions for the attacks to succeed:

1. every $x \in \mathcal{X}$ appears in at least one record (so the sets $\mathcal{S}_x$ are non-empty), also referred to as "density" in [NKW15, KKNO16];

2. the range queries $[x, y]$ are generated uniformly at random.

One benefit of assuming uniformly random range queries is that it makes our work directly comparable to Kellaris *et al.*'s [KKNO16]. However, as already discussed, our full reconstruction algorithms are data-optimal. As a consequence these algorithms consume the minimum possible amount of data for any query distribution. Thus the uniformity assumption plays no role in the design of the algorithm. Instead the assumption is used to compute data upper bounds. As we also show, very similar bounds would be obtained for the distribution where left end points are uniformly random; more generally, we conjecture that similar bounds would be achieved for a wide range of "non-pathological" distributions. (This is by contrast to *e.g.* Kellaris *et al.*'s attack in the non-dense case, which relies on a statistical inference approach that directly exploits the expected distribution of range queries.)

As intuition for the attack when rank leakage is available, consider the simpler situation where the x values appearing in the queried ranges $[x, y]$ are uniformly random.[1] Then with $\Omega(N \log N)$ queries, with high probability, every possible x value occurs at least once. This is a consequence of the standard analysis of the coupon collector's problem. The different x values are easily identified since they correspond to different values of $\mathsf{rank}(x-1)$ and the latter leaks to the adversary. It is then possible to recover the sets $\mathcal{S}_x$ with some simple set manipulations. What we show is that this simple example can be refined into a data-optimal algorithm that encompasses the case of uniformly random queries; and, perhaps surprisingly, that it can be extended to the generic setting where only access pattern leakage is available.

As a concrete example, if the values x represent age in years and we have $N = 125$, then our attack reduces by a factor of about 100 the number of queries that need to be observed for complete reconstruction, as compared to [KKNO16]. Specifically, their attack requires an expected number of queries $\frac{N(N+1)}{2} H_{N(N+1)/2} \approx 75,196$ queries. On the other hand, the expected number of queries for full reconstruction with our attack is at most $(N+1)(\log(N)+1) + 8\sqrt{N} + 6 \approx 830$ queries, or, if rank leakage is available, $N(\log(N)+1) + 4.4\sqrt{N} + 4 \approx 782$ queries (here we use the tighter bounds from Propositions 1 and 2). Thus the gap is significant even for relatively low values of N.

Approximate reconstruction attack. In some situations, exactly recovering the value x associated with each record is not necessary, and it may be sufficient to approximately learn these values. For example, the values may pertain to age or salary, and for the attacker, learning only an approximation may be good enough.

In this situation, we show that, under the same assumptions as our first attack, $O(N)$ queries suffice, even in the general setting where only the access pattern is leaked. More precisely, within an expected number of $\frac{5}{4} N \log(1/\epsilon) + O(N)$ queries, the attacker can reconstruct the values x associated with every record with a maximum error of ϵN, for any arbitrary precision ϵ. In particular, the data requirement only grows logarithmically with the desired precision ϵ. Once again the hidden constants are quite reasonable: an exact upper bound is $(1+\epsilon/4)N\log(1/\epsilon) + (2+4\sqrt{\epsilon})N + 4/\epsilon$ for $N \geq 40$. We

[1] If $[x, y]$ is generated uniformly at random, by choosing one of the possible intervals with equal probability, then x is not uniformly random, but rather weighted towards low values, with, for example $x = N$ appearing with probability only $2/(N(N+1))$.

also prove a lower bound $N \log(1/\epsilon)/2 - O(N)$ on the expected number of queries before *any* algorithm can achieve approximate reconstruction with precision ϵN.

The intuition behind this improvement over the full reconstruction attack is that, as discussed, our first attack ultimately reduces to solving a variant of the coupon collector's problem on the set $[1, N]$, with the x-values in the range queries $[x, y]$ playing the role of coupons. However, collecting the last few coupons is relatively expensive, and most of the coupons can be procured with $O(N)$ queries. Moreover, collecting most of the coupons is sufficient to approximately reconstruct the values associated to all records.

Going back to the example where $N = 125$, the approximate reconstruction attack starts to overtake the full reconstruction attack (without rank information) for $\epsilon \approx 0.05$, *i.e.* when reconstructing records up to a precision of 5%. Per our definition of precision, this means that for all records, the attacker has successfully determined an interval of size $0.05 \cdot N$ containing the value of the record; in particular, if the attacker were to guess the value of each record as the middle of the corresponding interval, then their guess would never be wrong by more than 2.5% of N. For a precision of 10% (which allows estimating the value of all records up to 5%), the required number of queries is at most 744, compared to 830 for full reconstruction. (Of course this gap grows with N: for $N = 10000$, the ratio of the expected number of queries for the former attack compared to the latter is about half.)

Inference attack using an auxiliary distribution. Recent research has shown that an attacker equipped with an auxiliary (or reference) distribution for the plaintext can wreak havoc against deterministic encryption schemes in a snapshot attack model by using basic frequency analysis [NKW15]. The assumption that an auxiliary distribution is available is a mild one in practice, for it is often the case that the type of data in a given column in an encrypted database is known or can be inferred from the diversity of its values [NKW15], and a rough and ready auxiliary distribution can be obtained from public statistics.

In our third attack, we show how an auxiliary distribution can be combined with rank and access pattern leakage to perform even better reconstruction attacks. The main insight is that the rank leakage from each query can be converted into information about the position of individual records in the ordered list of all records (such a list is intrinsic to schemes like Arx). By analysing the appearances of each record in response to multiple queries, the possible range of positions in which that record lies can be gradually constrained. The final range for that record is then mapped back to a small set of possible values using the inverse CDF of the auxiliary distribution. We show how to exploit these ideas to obtain more accurate attacks with fewer queries. Because this third attack is not amenable to a rigorous analysis, we resort to an empirical evaluation. In particular, we study the relationship between number of random range queries available and the accuracy of reconstruction of "age in years" data, extracted from a real-world medical dataset. In our experiments, observing only 50 queries was sufficient to reconstruct 95% of one hospital's records to within 10 years, 55% of its records to within 5 years, and 35% of its records to within 3 years. With more than 100 queries, our experimental results were limited only by the accuracy of the auxiliary distribution.

1.3 Applications

To the best of our knowledge, all practical encryption schemes supporting range queries leak access pattern information. This makes our generic attacks requiring only access pattern leakage widely applicable. However, it should be noted that schemes such as [BLR+15, CLWW16] that rely on order-preserving or order-revealing encryption (OPE/ORE) are already vulnerable to powerful statistical inference attacks by snapshot adversaries, such as [NKW15].[2]

As a consequence, our attacks are more relevant to second-generation schemes that attempt to reduce or eliminate leakage against snapshot adversaries. These schemes can be divided into two groups: schemes that leak access pattern but not rank, and those that also leak rank information. The

[2] In particular, any ORE (*a fortiori* OPE) scheme in the strict sense, *i.e.* where there exists a public algorithm enabling order comparison on ciphertexts, is trivially broken whenever the database is dense, as ordering all ciphertexts allows them to be directly matched with plaintext values.

first group includes Blind Seer by Pappas *et al.* [PKV+14], as well as a scheme by Faber *et al.* [FJK+15]; they also include approaches based on Range Predicate Encryption, such as [Lu12]. These schemes are vulnerable to our full and approximate reconstruction attacks.

The second group includes a variety of recent schemes [LW16, YGW+17, PBP16, Ker15, RACY16, HK16, AEJ+15]. In the remainder of this section, we look at some of these schemes in more detail and explain how they leak rank information. Since all of the schemes listed below leak rank information, they are vulnerable to all of our attacks. In addition to the schemes that are listed below, the recent POPE scheme by Roche *et al.* [RACY16], and a scheme with similar leakage by Hahn and Kerschbaum [HK16], should also be mentioned: although these two schemes mainly target scenarios where the database is not dense, our third attack remains applicable.

Lewi-Wu scheme. At CCS 2016, Lewi and Wu introduced a new practical construction for order-revealing encryption [LW16]. This construction also serves as the basis for supporting range queries in EncKV [YGW+17]. The Lewi-Wu scheme is not strictly an order-revealing scheme; in particular, it is fully secure against a basic snapshot adversary. One of the key ideas of the Lewi-Wu scheme is to allow encryption of plaintext values into two types of ciphertexts, named left and right ciphertexts. The database itself only contains *right* ciphertexts, which are semantically secure. Meanwhile range queries are handled by sending to the server *left* ciphertexts for both ends of the range. Left ciphertexts can be compared (in the sense of order comparison) with right ciphertexts using a public algorithm, which allows the server to identify values in-between the two end points of the range. However in this process, the server also learns which values are below the lower end point, and in particular the number of these values; and likewise for values above the higher end point. Thus rank information is leaked to the server.

Arx. Arx is a second-generation scheme recently proposed by Poddar *et al.* [PBP16]. In the previous reference, the authors mention ongoing collaboration with health data cloud provider Chino and European medical project UNCAP to deploy their scheme. Amongst other features, Arx builds a special Arx-Range index [BPP16] to support efficient range queries on encrypted data. Such an index is built for each field that will need to be ordered or queried by range.

The security of Arx is proven in terms of leakage; security theorems establish that the scheme does not leak more than is captured by the leakage functions as observed by a persistent, honest-but-curious adversary playing the role of the server [PBP16, Appendix A]. In general, the leakage of a query is the type of query, the time it was issued, and which records or fields are accessed, but not their content, nor any constants in the query. For a range query on interval $[x, y]$, the Arx-Range paper [BPP16, Theorem 8.1] states that the rank of the values $x - 1$ and y is leaked in addition to the access pattern $\mathcal{S}_{[x,y]}$. In particular, the values of x and y are not (directly) leaked by the range queries, and, being encrypted, the contents of each matching record are not leaked. The Arx-Range index does actually leak the rank and identifier set, so the leakage model is exact, and our attacks apply.

FH-OPE. Another example of a second-generation scheme is Kerschbaum's frequency-hiding, order-preserving encryption (FH-OPE) scheme [Ker15]. This scheme ensures that the same plaintext is never encrypted twice to the same ciphertext, thus thwarting basic frequency analysis attacks for a snapshot attacker (cf. [NKW15]), but also preserves order, so that range queries are easy to perform on encrypted data. At a high level, the scheme operates as follows. The client's state records the possible range of encrypted values for each plaintext value. Using this state, the client can rewrite a query for values in the range $[x, y]$ to one for values in the range $[\min(x), \max(y)]$, where $\min(z)$ and $\max(z)$ are bounds on the minimum and maximum size of ciphertexts corresponding to the value z. This scheme can be cast in the same terminology as we have used above, with FH-OPE ciphertexts corresponding to records (and their identifiers) and values x corresponding to plaintexts. It is then immediately apparent that the scheme leaks rank information. This is because queries leak the values of $\min(x)$ and $\max(y)$ to a persistent, honest-but-curious adversary playing the role of the server, and this adversary can always compute exact information about the ranks of x and y by comparing $\min(x)$ and $\max(y)$ to the complete set of (ordered) ciphertexts.

Cipherbase. Our attacks also apply to Cipherbase [AEJ+15], an encrypted database system that uses a combination of custom equality and range indices and trusted hardware. The range index is a B-tree, whose nodes each contain a number of encrypted value-record ID pairs, sorted by value. Traversing the tree is done by performing a series of order comparisons, outsourced to a trusted module, which decrypts and compares two values. By observing the access pattern as the tree is traversed to find the endpoints of a range query, an attacker learns the set of matching record IDs (by looking at those between the two endpoints) and the ranks of two values (by counting how many values are to the "left" of the endpoints in the B-tree). Thus, even though the scheme uses IND-CPA encryption and a secure hardware module, the sorted nodes in the range index and the access pattern provide enough leakage for our attacks.

For all of the aforementioned schemes except FH-OPE, the only attacks we are aware of beside ours is that of Kellaris *et al.* [KKNO16], which mainly focuses on the non-dense setting, and requires a significantly larger number of queries. In the non-dense case, the approach of Kellaris *et al.* is to use statistical inference based on the uniformity of queries; in the dense case, they collect all $O(N^2)$ possible queries, resulting in a simple sorting step.

In the particular case of Kerschbaum's FH-OPE scheme [Ker15], an attack was already mounted by Grubbs *e*t al. [GSB+17]. Their attack is in the snapshot setting and thus did not use query leakage. Attacks of a similar nature against OPE and ORE schemes can also be found in [DDC16]. Although our attacks require query leakage, they are more generic, as the scheme is only required to leak access patterns (and in some cases rank information). Another difference is that the attack by Grubbs *et al.* requires knowledge of an auxiliary distribution, whereas our first two attacks (full and approximate reconstruction) do not.

2 The Full Reconstruction Attack

The purpose of a full reconstruction attack is to determine the sets $\mathcal{S}_x$ for all $x \in \mathcal{X}$. In other words, a full reconstruction attacks recovers the value associated with every record.

2.1 Setup of the Attack and Layout of the Section

We assume that an honest user of the system makes a sequence of Q randomly generated range queries $[x, y]$, each query returning a set of records $\mathcal{M} = \mathcal{S}_{[x,y]}$. Since we are in the persistent, honest-but-curious setting, the set of matching records $\mathcal{M}$ is learned by the adversary – this is the *access pattern* leakage.

In Section 2.2, we start with a simple introductory example. In Section 2.3, we focus on the case where range queries leak *rank information*, as defined in Section 1.1. In Section 2.4, we show how the attack can be extended to work with only access pattern leakage. Throughout this section, we assume that the database is dense, *i.e.* every value appears in at least one record (cf. Section 1.1).

In all cases, we prove that $N \log N + O(N)$ queries are enough for full reconstruction (and provide concrete bounds). In Appendix C, we prove a matching lower bound. An important note is that, throughout this section, we give algorithms that favour simplicity over efficiency. In Appendix A, we show how these algorithms can be made (much) more efficient.

2.2 A Simple Example

As a warm-up, we begin with an example that provides some intuition for why $N \log N + O(N)$ queries are enough for full reconstruction in the presence of rank information leakage. For simplicity, in the scope of this example, we assume that the left end points x of range queries $[x, y]$ are uniformly random.

In the rest of the article, this will not be the case, because we will instead assume that range queries are uniformly random (which implies that left end points are biased toward lower values).

Recall that each query on an interval $[x, y]$ returns the set of record identifiers whose value lies in $[x, y]$, that is:

$$\mathcal{S}_{[x,y]} := \{r \in \mathcal{R} : \mathsf{val}(r) \in [x, y]\}.$$

We let $\mathcal{M}_k = \mathcal{S}_{[x_k,y_k]}$ denote the set of identifiers returned for the k-th query on range $[x_k, y_k]$. Also recall that $a_k = \mathsf{rank}(x_k - 1)$, $b_k = \mathsf{rank}(y_k)$ denote the rank information leaked by the k-th query. For each query on range $[x_k, y_k]$, the adversary is able to observe the leakage $(a_k, b_k, \mathcal{M}_k)$, but not directly x_k or y_k.

Success condition. The following algorithm will succeed iff every value $x \in [1, N]$ appears as the left end point of some query. Because we assumed uniformly random left end points, the expected number of queries until this condition is satisfied is exactly the standard coupon collector's problem. As such, using standard arguments, it is equal to:

$$N \cdot H_N \leq N \left(1 + \int_{x=1}^{N} \frac{1}{x}\right) = N(1 + \log N)$$

where H_N is the N-th harmonic number. We now assume that the condition is satisfied: every left end point occurs in at least one query.

The algorithm. By reordering and relabelling, we may assume that the a_k's are in increasing order (so $a_{k+1} \geq a_i$ for each i). Because every x value occurs in at least one query, there are exactly N distinct a_k values, each corresponding to one of the N different entries in $\mathcal{X}$; in particular we must have $a_1 = 0$, corresponding to value 1. The adversary selects any N-subset of indices k_i, where $1 \leq i \leq N$, such that the a_{k_i} are all distinct. It then computes:

$$\mathcal{E}_i = \mathcal{M}_{k_i} \setminus \left(\bigcup_{\ell \geq k_{i+1}} \mathcal{M}_\ell \right), \quad 1 \leq i \leq N$$

where we define the union $\cup_{\ell \geq k_{i+1}} \mathcal{M}_\ell$ for $i = N$ to be the empty set.

It is not hard to see that the set $\mathcal{E}_j$ is precisely the set of record identifiers for records containing value $j \in \mathcal{X}$, that is, $\mathcal{E}_j = \mathcal{S}_j$. The idea is that $\mathcal{M}_{k_j}$ certainly contains all such records, but possibly also additional records (because the range of query k_j contains j but possibly other values too). Performing a set subtraction with the sets of record identifiers $\mathcal{M}_\ell$ for $\ell \geq k_{j+1}$ removes the additional records.

This concludes our description of the attack: within an expected number of queries $N \cdot H_N \leq N \log(N) + N$, all left end points are collected; and we have just seen that as a result, the value of all records can be recovered. Despite its simplicity, this example captures a large part of the intuition behind our attacks in this section and the next. In particular, the data complexity bounds in the following sections are ultimately based on reductions to (increasingly intricate) variants of the coupon collector's problem.

2.3 The Full Reconstruction Attack with Rank Information

Before moving on to full reconstruction attacks *without* rank information, we first present a more robust algorithm for full reconstruction in the presence of rank information leakage, which improves on the algorithm from the previous section in various ways. The general approach of this new algorithm, and in particular the notion of *partition of records*, will form the basis of later algorithms and their analysis. This new algorithm is also *data-optimal*, in the sense that if the algorithm fails, then full reconstruction is impossible for any algorithm given the input queries. As a consequence, this algorithm requires the minimum possible number of queries for *any* distribution of queries.

In particular, it must perform at least as well as the algorithm from the previous section in terms of the required number of queries, so it succeeds within an expected number of queries $\leq N(\log(N)+1)$ in the case of uniform left end points. Moreover we will show that it also succeeds with an expected number of queries $\leq N(\log(N)+2)$ for uniform queries (contrary to the algorithm from the previous section, which requires $\Omega(N^2)$ queries in that setting[3]).

The algorithm. The idea of the algorithm is as follows. As before, fix a set of Q queries on ranges $[x_k, y_k]$, $k \leq Q$. The corresponding leakage observed by the adversary is $\{(a_k, b_k, \mathcal{M}_k) : k \leq Q\}$, where $a_k = \mathsf{rank}(x_k - 1)$, $b_k = \mathsf{rank}(y_k)$, and $\mathcal{M}_k = \mathcal{S}_{[x_k,y_k]}$ is the set of records with values in $[x_k, y_k]$.

Define the following equivalence relation on records (recall that records are assimilated with their IDs). Two records are equivalent iff for all Q queries, either both records are matched by the query, or neither. At least intuitively, it is clear that two records can only be meaningfully distinguished given the set of queries iff they are in distinct equivalence classes. Moreover, the equivalence class of a record r can be easily computed as:

$$P(r) = \left(\bigcap_{\{k : r \in \mathcal{M}_k\}} \mathcal{M}_k \right) \Big\backslash \left(\bigcup_{\{k : r \notin \mathcal{M}_k\}} \mathcal{M}_k \right).$$

Since $\mathcal{M}_k = \{r \in \mathcal{R} : \mathsf{val}(r) \in [x_k, y_k]\}$, $P(r)$ contains exactly those records whose value lies in:

$$V(r) = \left(\bigcap_{\{k : r \in \mathcal{M}_k\}} [x_k, y_k] \right) \Big\backslash \left(\bigcup_{\{k : r \notin \mathcal{M}_k\}} [x_k, y_k] \right).$$

Note that neither $P(r)$ nor $V(r)$ can be empty, as they must contain at least, respectively, r and $\mathsf{val}(r)$. Note also that $V(r)$ is unknown to the adversary at this point in the attack.

To sum up, the set of $P(r)$'s (resp. $V(r)$'s) forms a partition of the set of records (resp. values), and each $P(r)$ contains exactly those records whose value lies in $V(r)$. Due to this bijection, the *partition of records* $\mathcal{P} = \{P(r) : r \in \mathcal{R}\}$ cannot contain more than N elements. On the other hand, since records in the same class cannot be meaningfully distinguished, full reconstruction requires $|\mathcal{P}| = N$.

The crucial point is that, conversely, the condition $|\mathcal{P}| = N$ is enough to enable full reconstruction. Indeed, if $|\mathcal{P}| = N$, then by the bijection with the $V(r)$'s, each $V(r)$ must contain a single value, and $P(r)$ constitutes the set of records having that value. Thus each element of $\mathcal{P}$ corresponds to a single value: $\mathcal{P} = \{\mathcal{S}_x : x \in \mathcal{X}\}$. Hence, in order to find the value of every record, it only remains to sort the elements of $\mathcal{P}$ by value. But this can be done using rank as a kind of proxy for value. Indeed, the value of $P(r)$ has rank

$$m(r) = \max\left(\left(\bigcap_{\{k : r \in \mathcal{M}_k\}} [a_k, b_k] \right) \Big\backslash \left(\bigcup_{\{k : r \notin \mathcal{M}_k\}} [a_k, b_k] \right) \right).$$

Then it suffices to sort all values $m(r)$ to correctly sort the corresponding classes $P(r)$ by value. Our algorithm does exactly this. Pseudo-code is provided in Algorithm 1. In this algorithm, we immediately compute the map m, and check that $\{m(r) : r \in \mathcal{R}\} = N$. First computing P and/or V is not necessary in this setting, but is only used above as an aid to understanding the algorithm.

Analysis of the algorithm. We make several claims about Algorithm 1. The first is that it is data-optimal, *i.e.* if the algorithm fails, then no algorithm can achieve full reconstruction with probability 1 given the same queries as input. The rationale we have provided for the algorithm already gives strong evidence that this is the case (which is to say, $|\mathcal{P}| = N$ is a necessary condition); however a more formal treatment is provided in Appendix D. Second, we observe that Algorithm 1 still achieves

[3] For the uniform distribution $\mathcal{U}$ on ranges, left end points within $[N-c, N]$ for any fixed constant c, only occur with probability $\Theta(1/N^2)$ – cf. Lemma 1 in Appendix B.1 for an exact distribution.

Algorithm 1 Full reconstruction attack with rank information.

```
FULL-RECONSTRUCTION($\mathcal{Q}$):
Input: set of queries $\mathcal{Q} = \{(a_k, b_k, \mathcal{M}_k)\}$.
Output: $\perp$, or map $\mathrm{Val} : \mathcal{R} \to \mathcal{X}$ s.t. $\forall r$ $\mathrm{Val}(r) = \mathsf{val}(r)$.
$m \leftarrow$ empty map, $\mathrm{Val} \leftarrow$ empty map
for all $r \in \mathcal{R}$ do                                                ▷ Partitioning step
    $m(r) \leftarrow \max\left(\left(\bigcap_{\{k:r\in\mathcal{M}_k\}}[a_k, b_k]\right) \setminus \left(\bigcup_{\{k:r\notin\mathcal{M}_k\}}[a_k, b_k]\right)\right)$
end for
$M \leftarrow \{m(r) : r \in \mathcal{R}\}$
if $|M| < N$ then
    return $\perp$
end if
Sort $M$ in increasing order                                                   ▷ Sorting step
for all $r \in \mathcal{R}$ do
    $\mathrm{Val}(r) \leftarrow$ index of $m(r)$ in sorted $M$                 ▷ Counting from 1
end for
return Val
```

full reconstruction with an expected number of queries upper-bounded by $N(\log(N)+1)$ in the setting where left end points are uniformly random. This is a direct consequence of data optimality and the observations in Section 2.2.[4]

More importantly, we claim that, unlike the algorithm from Section 2.2, Algorithm 1 achieves a similar data complexity in the case where range queries are uniformly random. More precisely, we show that for $N \geq 27$, the expected number of queries is upper-bounded by $N(\log(N)+2)$. For simplicity, in the following statement, we assume that N is a multiple of 4; it is clear that the success probability of the algorithm, and the expected number of queries before it succeeds, respectively increase and decrease with N, so we can always round N to the next multiple of 4 if necessary.

Proposition 1. *Assume N is a multiple of 4. Assume that the database is dense, and range queries are drawn uniformly at random. Then the probability of success of Algorithm 1 after Q queries is lower-bounded by:*

$$1 - 2e^{-Q/(2N+2)} - Ne^{-Q/N}.$$

Moreover, the expected number of queries before the algorithm succeeds is upper-bounded by:

$$N\log(N) + O(N).$$

Concretely, for $N \geq 27$, it is upper-bounded by $N(\log(N)+1) + 4.4\sqrt{N} + 4 \leq N(\log(N)+2)$.

A proof of Proposition 1 is given in Appendix B.2. As a concrete example, setting $N = 100$ yields an expected number of queries upper-bounded by 609 to achieve full reconstruction, regardless of the number of records, as long as the database is dense (*i.e.* every value occurs at least once). We emphasise that full reconstruction means recovering the value of every single record.

Matching lower bound. In Appendix C, we investigate whether $\Omega(N\log N)$ queries are in fact necessary for full reconstruction. Corollary 1 answers in the positive, showing that the expected number of queries for *any* algorithm to achieve full reconstruction is $\frac{1}{2}N\log(N) - O(N)$.

[4] Indeed data optimality implies that if for some distribution $\mathcal{D}$ on the set of queries, there exists an algorithm that achieves full reconstruction with an expected number of queries E, then for the same distribution the expected number of queries necessary for Algorithm 1 to succeed is at most E.

Complexity. For clarity, Algorithm 1 is stated in a way that closely follows the rationale exposed earlier in this section. However it is quite inefficient: line 3 in particular results in a large number of redundant computations, each involving multiple intersections and unions of potentially large sets. Keep in mind that in a real-world database, while the number of values N may be small, and we already know that $Q \approx N \log N$ queries are enough for full reconstruction, the number of records R can be quite large. In Appendix A, we show a very efficient approach to computing the partition of records, without explicitly computing any set intersections or unions. This results in a time complexity $O(Q(N+R))$, with little overhead over simply reading all queries.

2.4 The Full Reconstruction Attack with only Access Pattern Leakage

In the previous section, we studied full reconstruction in the presence of rank information leakage. We now extend the attack to the case where only the *access pattern* is leaked. That is, for each query, only the set of IDs for records matching the query is leaked to the adversary, and nothing else. Kellaris *et al.* have already shown that $O(N^2 \log N)$ queries are enough to achieve full reconstruction for dense databases [KKNO16]. However, we show that $N \log(N) + O(N)$ queries are still enough. As before, our assumptions are that range queries are uniformly distributed, and that the database is dense. The attack requires no a priori knowledge about the distribution of values among records.

A small caveat is that in the absence of rank information, full reconstruction can be achieved only up to *reflection*. By reflection, we mean the permutation of values that swaps value i with value $N+1-i$. The point is that if we compose the mapping val from records to values with this reflection, and also apply it to the end points $[x, y]$ of range queries (which preserves the uniform distribution), all else remaining equal, then access pattern leakage is unchanged. As a result, if we make no assumption about the distribution of values among records, an adversary whose view is restricted to access pattern leakage can only hope to recover the mapping val of records to values up to reflection. The same is of course true for the attacks in [KKNO16]. As a consequence, whenever we talk about (full) reconstruction with only access pattern leakage, we mean full reconstruction up to reflection (which, on the other hand, is possible, as we shall see).

The algorithm. Recall that the adversary only has access to queries on ranges $[x_k, y_k]$, and only sees the access pattern leakage $\mathcal{M}_k = \mathcal{S}_{[x_k, y_k]}$. In the remainder we identify a query with the corresponding set of matching records. We note that the algorithm can fail at a few points; however Proposition 7 in Appendix D shows that the algorithm is data-optimal, *i.e.* whenever it fails, although this may not be immediately apparent, full reconstruction was in fact impossible. The attack can be divided into a *partitioning* step and a *sorting* step.

The partitioning step is essentially identical to the attack from the previous section. Indeed, we can compute the partition of records $\mathcal{P}$ as introduced in Section 2.3:

$$\mathcal{P} = \left\{ \left(\bigcap_{\{k : r \in \mathcal{M}_k\}} \mathcal{M}_k \right) \Big\backslash \left(\bigcup_{\{k : r \notin \mathcal{M}_k\}} \mathcal{M}_k \right) : r \in \mathcal{R} \right\}.$$

Once again, observe that full reconstruction requires $|\mathcal{P}| = N$. Conversely if $|\mathcal{P}| = N$ every element of $\mathcal{P}$ is the set $\mathcal{S}_x$ of records with value x for some $x \in \mathcal{X} = [1, N]$. We will call the elements of $\mathcal{P}$ *points* (though the reader should keep in mind that each point is in general a set of records). Each point p corresponds to a distinct value in $\mathcal{X}$ (*viz.* the singleton $\mathsf{val}(p)$).

To achieve full reconstruction, it remains to assign the correct value to each point. This is equivalent to sorting $\mathcal{P}$ according to the value of each point. This is the *sorting* step. In the presence of rank information, the sorting step was essentially trivial. With only the access pattern, we proceed as follows. First, we set out to find an end point, *i.e.* a point with value 1 or N. Due to the reflection symmetry mentioned earlier, these two cases cannot be distinguished.

In order to find an end point, we form a maximal union S of queries that does not cover the full set of records, while ensuring that this union covers an interval of values. To ensure that the union of two queries covers an interval of values, it is enough to require that the two queries overlap (indeed the union of two overlapping intervals is an interval). Hence we build S as follows: we start with a query not covering the full set of records, and extend it for as long as possible with overlapping queries until we can no longer do so without covering the full set of records $\mathcal{R}$. The crux of the matter is that if $\mathcal{R} \setminus S$ is reduced to a single point, then it must be an end point. Indeed in that case S must cover an interval of $N - 1$ values within $[1, N]$, so the remaining value can only be 1 or N. If $\mathcal{R} \setminus S$ is not a single point, the algorithm fails.

Once an end point is identified, we assume that it corresponds to value 1 (which must be true up to reflection). We have thus determined the initial segment of points $I(1)$ containing a single point. We then propagate this information by building each initial segment of points $I(i) = \mathcal{S}_{[1,i]}$ in turn, by induction. Given an initial segment $I(i)$, this amounts to finding the *next* point, which must be $\mathcal{S}_{i+1}$. To do this, we build the intersection T of all queries overlapping $I(i)$ and containing at least one point outside $I(i)$. We then subtract from T all queries overlapping T from the right (*i.e.* overlapping T and not contained in $I(i) \cup T$) so long as T remains non-empty. If the resulting T contains a single point, it must be $\mathcal{S}_{i+1}$; we let $I(i + 1) = I(i) \cup T$ and continue. Otherwise the algorithm fails.

Once we have built the last (proper) initial segment $I(N-1)$, we have successfully sorted all points, and hence determined the value of all records. Pseudo-code is provided in Algorithm 2.

Analysis of the algorithm. The properties of Algorithm 2 are similar to those of Algorithm 1 in the previous section. The algorithm is data-optimal: if it fails on some input, then full reconstruction (up to reflection) with probability 1 is impossible for any algorithm given that input. A proof is provided in Appendix D. More importantly, the expected number of queries necessary for the algorithm to succeed in achieving full reconstruction, given uniformly random ranges queries, is $N \log(N) + O(N)$. More precisely, the following holds.

Proposition 2. *Assume N is a multiple of 4. Assume that the database is dense, and range queries are drawn uniformly at random. Then the probability of success of Algorithm 2 after Q queries is lower-bounded by:*

$$1 - 4e^{-Q/(2N+2)} - Ne^{-Q/(N+1)}.$$

Moreover, the expected number of queries before the algorithm succeeds is upper-bounded by:

$$N \log(N) + O(N).$$

Concretely, for $N \geq 26$, it is upper-bounded by $(N + 1)(\log(N) + 1) + 8\sqrt{N} + 6 \leq N(\log(N) + 3)$.

A proof of Proposition 2 is provided in Appendix B.3. Since the above proposition is one of our main results, we provide here some intuition regarding why $N \log(N) + O(N)$ queries is still enough, even without rank information. For simplicity, we go back to the setting of the introductory example from Section 2.2, and assume that the left end points of range queries are uniformly random. The reasoning from Section 2.2 already shows that if all N possible left end points appear in some range query, then $|\mathcal{P}| = N$. Thus we have N *points* (elements of $\mathcal{P}$), and the partitioning step succeeds. It remains to sort points.

For the sorting step to succeed, the previous condition is not enough. For example, if all values $x \in [1, N]$ were to appear merely in singleton range queries $[x, x]$, then we would indeed have N points, but would be thoroughly unable to sort them. Fortunately, the main idea underlying the proof of Proposition 2 is that such singleton sets are in some sense the main obstacle – in fact, as long as $|\mathcal{P}| = N$, it is rather unlikely that query leakage does not provide enough information to be able to sort points.

More precisely, if we strengthen the previous condition very mildly by requiring that (1) all left end points $< N$ appear in some range query on an interval of length at least 2 (so singletons are discarded); and (2) there exists a range query of the form $[x, N - 1]$ for some $x < N - 1$; then with these two conditions it is straightforward to check that Algorithm 2 cannot fail at lines 14 or 24, and hence the sorting step succeeds.[5]

[5] This point is not developed as that would be redundant with the real proof.

Algorithm 2 Full reconstruction attack with only access pattern.

```
Full-Reconstruction-AP($\mathcal{Q}$):
Input: set of queries $\mathcal{Q} = \{\mathcal{M}_k\}$.
Output: $\perp$, or map $\mathrm{Val} : \mathcal{R} \to \mathcal{X}$ s.t. $\forall r\ \mathrm{Val}(r) = \mathsf{val}(r)$ or $\forall r\ \mathrm{Val}(r) = N + 1 - \mathsf{val}(r)$.
$P \leftarrow$ empty map, $I \leftarrow$ empty map, $\mathrm{Val} \leftarrow$ empty map
for all $r \in \mathcal{R}$ do                                              ▷ Partitioning step
    $P(r) \leftarrow \left(\bigcap_{\{k:r\in\mathcal{M}_k\}} \mathcal{M}_k\right) \setminus \left(\bigcup_{\{k:r\notin\mathcal{M}_k\}} \mathcal{M}_k\right)$
end for
$\mathcal{P} \leftarrow \{P(r) : r \in \mathcal{R}\}$
if $|\mathcal{P}| < N$ then
    return $\perp$
end if
Pick $S \in \mathcal{Q}$ s.t. $|S| < R$                                        ▷ Sorting step
while $\exists q \in \mathcal{Q}$ s.t. $q \cap S \neq \varnothing,\ q \setminus S \neq \varnothing,\ q \cup S \neq \mathcal{R}$ do      ▷ Searching for end point
    $S \leftarrow S \cup q$
end while
if $\mathcal{R} \setminus S \notin \mathcal{P}$ then                          ▷ Ensuring $|\mathsf{val}(\mathcal{R} \setminus S)| = 1$
    return $\perp$
end if
$I(1) \leftarrow \mathcal{R} \setminus S$                                     ▷ Found end point
for all $i \in [1, N-1]$ do                                              ▷ Searching for the next point
    $\mathcal{Q}' \leftarrow \{q \in \mathcal{Q} : q \cap I(i) \neq \varnothing, q \setminus I(i) \neq \varnothing\}$
    $T \leftarrow \left(\bigcap_{q\in\mathcal{Q}'} q\right) \setminus I(i)$
    while $\exists q \in \mathcal{Q}$ s.t. $q \cap T \neq \varnothing, q \setminus (T \cup I(i)) \neq \varnothing, T \setminus q \neq \varnothing$ do
        $T \leftarrow T \setminus q$
    end while
    if $T \notin \mathcal{P}$ then                                          ▷ Ensuring $|\mathsf{val}(T)| = 1$
        return $\perp$
    end if
    $I(i+1) \leftarrow I(i) \cup T$                                         ▷ Found next point
end for
for all $r \in \mathcal{R}$ do                                              ▷ Success
    $\mathrm{Val}(r) \leftarrow \min\{i : r \in I(i)\}$
end for
return Val
```

In the proof of Proposition 2 (Appendix B.3), the actual requirements are slightly more intricate. This is because range queries are assumed to be uniformly distributed, which implies that neither left nor right end points are uniformly distributed (instead, their distribution is close, respectively, to the min and max of two uniform values). However the underlying idea is the same.

Complexity. As already mentioned, our algorithms are described in an effort to maximise legibility and ease of analysis, rather than efficiency. However in Appendix A, we show that Algorithm 2 can be executed efficiently, in time complexity $O(Q(N^2 + R))$.

3 The Approximate Reconstruction Attack

In the previous section, we showed that $N \log(N) + O(N)$ randomly distributed range queries are sufficient to recover the exact value x contained in every record, a full reconstruction attack. However, if we are content with recovering the value associated with every record up to a margin of error, say,

ϵN for some fixed arbitrary $\epsilon > 0$, (*e.g.* if we wish to recover the value of all records within 1%), then we show in this section that the required number of queries is only $O(N)$. We limit our attention to the general setting where only access pattern leakage is available (a data-optimal variant when rank information is available is given in Appendix A).

More formally, an *approximate* reconstruction attack seeks to determine the value $\mathsf{val}(r)$ of every record r, up to some precision $k = \epsilon N$. That is, for every record r, an approximate reconstruction attack with precision k outputs an interval $[x, x+k]$ such that $\mathsf{val}(r) \in [x, x+k]$. A full reconstruction attack may be regarded as the special case where $k = 0$. In the remainder however, we shall assume $k > 0$: the case $k = 0$ requires special treatment, and was already covered in the previous section. Without loss of generality, we may assume that $k = \epsilon N$ is an integer. Note that our algorithm will either succeed, *i.e.* it correctly recovers all values up to precision ϵ as defined above; or it outputs $\perp$. (In other words, the algorithm can fail with $\perp$, but it cannot output an incorrect answer.)

The main result of this section is an approximate reconstruction attack for which the expected number of queries necessary for success is upper-bounded by $\frac{5}{4}N\log(1/\epsilon) + O(N)$. In particular, it is $O(N)$ for any fixed ϵ, and only grows logarithmically with the precision ϵ. We stress that our attack exploits only access pattern leakage and does not require rank leakage.

3.1 Intuition for the Approximate Reconstruction Attack

Before proving this result, we provide some intuition as to why the complexity drops from $O(N \log N)$ to $O(N)$ when a margin of error is allowed in a reconstruction attack.

The reason why recovering the value of every record required $O(N \log N)$ queries is because the problem essentially reduced to a coupon collector's problem on N items, as seen in Section 2.2. Each coupon was an integer in the interval $[1, N]$. After $k < N$ distinct coupons have been drawn, drawing a new coupon requires $N/(N-k)$ tries on average, since each new draw has a probability $(N-k)/N$ of yielding a new coupon, distinct from the k coupons already collected. It follows that the expected number of draws to gather all N coupons is:

$$\sum_{k=0}^{N-1} \frac{N}{N-k} = N \sum_{k=1}^{N} \frac{1}{k} = N \cdot H_N$$

where H_N is the N-th harmonic number.

A simple observation however is that the last few coupons are much more expensive to collect than the early ones, in terms of the expected number of draws required. Indeed, the last coupon requires N draws on average, the previous one $N/2$ draws, etc. In particular, if we only wish to recover $(1-\epsilon)N$ coupons for some $\epsilon > 0$, then the expected number of draws is:

$$\sum_{k=0}^{N-\epsilon N-1} \frac{N}{N-k} = N \sum_{k=\epsilon N+1}^{N} \frac{1}{k} < N \int_{\epsilon N}^{N} \frac{1}{x} = N \log\left(\frac{1}{\epsilon}\right)$$

where ϵN is assumed to be an integer. Thus, the expected number of tries is $O(N)$, with the growth being proportional to $\log(1/\epsilon)$.

3.2 Algorithm for the Approximate Reconstruction Attack

With the above intuition in mind, we now present an approximate reconstruction algorithm. As usual, we assume that the database is dense, and that range queries are uniformly distributed. No assumption is made about the a priori distribution of values among records. The input of the algorithm, *i.e.* the view of the adversary, is limited to access pattern leakage. Recall from Section 2.4 that with access pattern leakage, we can only recover record values up to *reflection*, *i.e.* we cannot distinguish between the real assignment of values $r \mapsto \mathsf{val}(r)$, and its reflection $r \mapsto N+1-\mathsf{val}(r)$. A fortiori, the algorithm also succeeds if rank information is available (and in that case, the limitation that values are recovered up to reflection disappears).

Compared to the full reconstruction algorithms from Section 2, particular care must be taken around a few issues. The main one is that, because the partition of records $\mathcal{P}$ at the core of the previous algorithms no longer satisfies $|\mathcal{P}| = N$, there is no guarantee that its elements correspond to *intervals* of values (as opposed to arbitrary subsets). As a result we eschew this approach, and instead ensure at every step that the subsets of records we build correspond to intervals of values (*i.e.* for every subset S of records built in the course of the algorithm, $\mathsf{val}(S)$ is an interval). Our main tool toward this purpose are two elementary observations: an intersection of intervals is an interval, and a union of overlapping intervals is an interval.

The algorithm. Suppose we are given access to a set of queries $\mathcal{Q} = \{\mathcal{M}_k : k \leq Q\}$ on ranges $[x_k, y_k]$, with $\mathcal{M}_k = \mathcal{S}_{[x_k,y_k]}$. Our strategy for approximate reconstruction proceeds in two steps.

(1) First, we want to split the set of records into two "halves", as follows. Let r be an arbitrary record (all possible choices of this record will be tried until the algorithm succeeds). Let M denote the intersection of all queries containing r. We wish to find two sets of records half_L and half_R such that $\text{half}_L \cup \text{half}_R$ contains all records, $\text{half}_L \cap \text{half}_R = M$, and finally both $\mathsf{val}(\text{half}_L)$ and $\mathsf{val}(\text{half}_R)$ are intervals. The point of this setup is that it partitions the set of records into three subsets: $\text{half}_L \setminus M$, M, and $\text{half}_R \setminus M$; such that the corresponding sets of values $\mathsf{val}(\text{half}_L \setminus M)$, $\mathsf{val}(M)$, and $\mathsf{val}(\text{half}_R \setminus M)$ is a partition of $[1, N]$ into three successive intervals. We will then sort records independently in half_L and half_R.

The exact technique used to build half_L and half_R is given in lines 3-6 of Algorithm 3. The idea is to build each subset as a union of two overlapping queries. This ensures that the corresponding value set is an interval, while incurring only a cost $O(N)$ in the expected number of queries for the algorithm to succeed.

(2) The next step is to sort records within $\text{half}_L \setminus M$. To do so, define a *left coupon* as a set of records of the form $q \setminus \text{half}_R$ such that q is a query (*i.e.* $q = \mathcal{M}_k$ for some k) containing M. We claim that the set $\mathcal{C}_L$ of left coupons is linearly ordered for $\subset$. To see this, shift the perspective from records to their values: let $[x_M, y_M] = \mathsf{val}(M)$; then every left coupon is equal to $\mathcal{S}_{[x,x_M-1]}$ for some $x < x_M$. Indeed, any query q containing M is such that $\mathsf{val}(q) = \mathcal{S}_{[x,y]}$ for some $x \leq x_M \leq y$, and $\mathsf{val}(\text{half}_R) = [x_M, N]$. The linear order $\subset$ on left coupons clearly implies the (reverse) order for the value of new records appearing in each successive coupon. Thus we have identified and sorted $n_L = |\mathcal{C}_L|$ distinct sets of records, all below x_M.

We repeat the process for half_R to partition half_R into $n_R = |\mathcal{C}_R|$ ordered subsets, all above y_M. Finally we have thus sorted $n_L + n_R + 1$ distinct subsets of records, whose union covers all records.[6] Since the values appearing in each subset of the global partition must be distinct, the values appearing in the k-th subset must be at least k; likewise they can be at most $k + N - (n_L + n_R + 1)$ (since the ℓ-th set counting from the right can contain values at most $N + 1 - \ell$). Hence if $n_L + n_R + 1 = (1 - \epsilon)N$, we have succeeded in approximate reconstruction with precision ϵN. If that condition is not satisfied, pick the next value of r at step (1) and try again.

Pseudo-code is provided in Algorithm 3.

Analysis of the algorithm. Contrary to Algorithms 1 and 2, Algorithm 3 is not data-optimal. However a data-optimal variant in the case where rank information is available is given in Appendix A.

The main feature of Algorithm 3 is expressed by the following proposition. As before, we assume for simplicity that N is a multiple of 4; note that the expected number of queries for the algorithm to succeed is monotone increasing in N, so this has little impact.

[6] Recall that we assume $[1, N]$ to be a query; or equivalently, that the set of all records is known. As a result $\mathcal{S}_{[1,x_M-1]}$, $\mathcal{S}_{[y_M+1,N]}$ must appear, respectively, as left and right coupons.

Algorithm 3 Approximate reconstruction attack with only access pattern.

```
APPROXIMATE-RECONSTRUCTION-AP($\mathcal{Q}$):
Input: set of queries $\mathcal{Q} = \{\mathcal{M}_k\}$, real $0 < \epsilon < 1$.
Output: $\perp$, or maps $\mathrm{minVal}, \mathrm{maxVal} : \mathcal{R} \to \mathcal{X}$ s.t. $\forall r, \mathsf{val}(r) \in [\mathrm{minVal}(r), \mathrm{maxVal}(r)]$ or $\forall r, \mathsf{val}(r) \in [N+1-\mathrm{maxVal}(r), N+1-\mathrm{minVal}(r)]$; and $\forall r, \mathrm{maxVal}(r) - \mathrm{minVal}(r) \leq \epsilon N$.
for all $r \in \mathcal{R}$ do
    $M \leftarrow \bigcap_{\{k : r \in \mathcal{M}_k\}} \mathcal{M}_k$                              ▷ Partitioning step
    Find $q_L, q_R$ s.t. $q_L \cap q_R = M$, maximizing $|q_L \cup q_R|$
    Find $q'_L$ s.t. $q'_L \cap q_L \neq \varnothing$, $q'_L \cap q_R \subseteq M$, maximizing $|q'_L \cup q_L|$
    Find $q'_R$ s.t. $q'_R \cap q_R \neq \varnothing$, $q'_R \cap q_L \subseteq M$, maximizing $|q'_R \cup q_R|$
    if $q'_L \cup q_L \cup q_R \cup q'_R = \mathcal{R}$ then
        $\mathrm{half}_L \leftarrow q'_L \cup q_L$
        $\mathrm{half}_R \leftarrow q_R \cup q'_R$
        $\mathcal{C}_L \leftarrow \{q \setminus \mathrm{half}_R : q \in \mathcal{Q}, M \subseteq q\} \setminus \{\varnothing\}$          ▷ Left sorting step
        $n_L \leftarrow |\mathcal{C}_L|$
        $(\mathcal{C}_L[1], \ldots, \mathcal{C}_L[n_L|) \leftarrow$ sort $\mathcal{C}_L$ for order $\subset$          ▷ It is a linear order
        $\mathcal{C}_R \leftarrow \{q \setminus \mathrm{half}_L : q \in \mathcal{Q}, M \subseteq q\} \setminus \{\varnothing\}$          ▷ Right sorting step
        $n_R \leftarrow |\mathcal{C}_R|$
        $(\mathcal{C}_R[1], \ldots, \mathcal{C}_R[n_R]) \leftarrow$ sort $\mathcal{C}_R$ for order $\subset$          ▷ It is a linear order
        if $N - (n_L + n_R + 1) \leq \epsilon N$ then
            for all $r \in \mathcal{R}$ do                                    ▷ Success
                if $r \in \mathrm{half}_L$ then
                    $\mathrm{minVal}(r) \leftarrow n_L + 1 - \min\{i : r \in \mathcal{C}_L[i]\}$
                else if $r \in M$ then
                    $\mathrm{minVal}(r) \leftarrow n_L + 1$
                else if $r \in \mathrm{half}_R$ then
                    $\mathrm{minVal}(r) \leftarrow n_L + 1 + \min\{i : r \in \mathcal{C}_R[i]\}$
                end if
                $\mathrm{maxVal}(r) \leftarrow \mathrm{minVal}(r) + N - (n_L + n_R + 1)$
            end for
            return minVal, maxVal
        end if
    end if
end for
return $\perp$
```

Proposition 3. *Assume N is a multiple of 4, $\epsilon N/2$ is an integer, and $\epsilon < 3/4$. Assume that the database is dense, and range queries are drawn uniformly at random. Then the probability of success of Algorithm 3 after Q queries is lower-bounded by:*

$$1 - 4e^{-Q/(2N+2)} - 2e^{-Q\log(1+\epsilon/2)+\epsilon N/2\cdot(\log(1/\epsilon)+1.5)}.$$

Moreover, the expected number of queries before the algorithm succeeds is upper-bounded by:

$$\frac{5}{4}N\log(1/\epsilon) + O(N).$$

Concretely, for $N \geq 40$, it is upper-bounded by $(1+\epsilon/4)N\log(1/\epsilon) + (2+4\sqrt{\epsilon})N + 4/\epsilon$.

A proof of Proposition 3 is given in Appendix B.4. As a corollary of the expected number of queries given above, if we want to recover the value of all records within a constant *additive* error, *i.e.* $\epsilon = \Theta(1/N)$, then $O(N\log N)$ queries suffice: as one might expect, this matches the full reconstruction attack. Another observation is that for fixed ϵ, $O(N)$ queries suffice; and perhaps surprisingly, the expected number of queries only grows logarithmically with the desired precision.

The formula for the probability of success may appear difficult to parse. It may help to observe that the last term is dominant for small ϵ. Furthermore, for the sake of providing some intuition, if we approximate $\log(1+\epsilon/2)$ by $\epsilon/2$, and disregard the final "+1.5", then the last term would dictate a probability of failure upper-bounded by $2/e < 3/4$ for $Q = N\log(1/\epsilon) + 2/\epsilon$ queries, and this upper bound would be divided by $e \approx 2.7$ for every additional $2/\epsilon$ queries. Once again if $\epsilon = \Theta(1/N)$ this matches the approximate behaviour of the full reconstruction attack.

Matching lower bound. In Appendix C, we investigate whether $\Omega(N\log(1/\epsilon))$ queries are in fact necessary for approximate reconstruction. Proposition 5 in Appendix C answers in the positive, showing that the expected number of queries before *any* algorithm can achieve approximate reconstruction with precision ϵN is at least $\frac{1}{2}N\log(1/\epsilon) - O(N)$ (where the constant in $O(N)$ is independent of ϵ).

Complexity. As was the case with Algorithms 1 and 2, Algorithm 3 was defined with legibility in mind, rather than efficiency. A more efficient variant is discussed in Appendix A, and achieves a time complexity $O(Q(N^2+R))$ with only access pattern leakage, and $O(Q(N+R))$ with rank information.

4 Exploiting Auxiliary Information

Up to this point, we have focussed on trying to recover the values associated with records, without making any assumptions about the distribution of those values. However, in a real-world scenario, this distribution may be quite predictable.[7] For instance, the values might represent the age of patients in a hospital or salaries in a personnel database. In this context, the auxiliary information provided by a known distribution of values can help predict the values of some records, even if relatively few range queries are available.

In this section, we propose a heuristic algorithm for performing reconstruction attacks against schemes that have rank leakage (such as Lewi and Wu's ORE scheme, Arx, and FH-OPE) in the setting where this auxiliary distribution is available. Thus our attack will now make use of rank leakage, access pattern leakage, and the auxiliary distribution. Our attack is not amenable to rigorous analysis, unlike our previous two attacks, so we will rely on an empirical evaluation of its performance.

While we introduce a new assumption about the availability of pertinent auxiliary information, we also remove an assumption that applied to previous attacks: we no longer assume the data is dense. So, while query end points are still sampled uniformly at random from $\mathcal{X}$, not all of these values necessarily correspond to a record or records. As in the previous attacks, we assume that the adversary knows the set of record identifiers $\mathcal{R}$ and we treat this set as $[1, R]$ without loss of generality.

4.1 The Algorithm

Setup. As usual, we consider the scenario where a client is issuing uniformly distributed range queries on ranges $[x_k, y_k] \subseteq \mathcal{X}$, for $k \leq Q$. For each query on range $[x_k, y_k]$, the adversary observes the leakage $(a_k, b_k, \mathcal{M}_k)$, where the *access pattern* leakage $\mathcal{M} = \mathcal{S}_{[x_k,y_k]} = \{r \in \mathcal{R} : \mathsf{val}(r) \in [x_k, y_k]\}$ contains the set of matching records IDs; and the *rank information* leakage $a_k = \mathsf{rank}(x_k - 1)$, $b_k = \mathsf{rank}(y_k)$ is such that $[a_k+1, b_k]$ can be interpreted as the positions of the records in $\mathcal{M}_k$ within a fixed list L of all records sorted by value (cf. Section 1.1).

The algorithm. Let us define the *position* of a record in L as follows: the position $\mathsf{pos}(r)$ of a record r is its position in the sorted list L (counting from 1). From this perspective, the rank of a value x is the position of the last record with value x. The algorithm proceeds in two steps. In Step 1, based on available queries, for each record r, we compute an interval $[a, b]$ such that $\mathsf{pos}(r) \in [a, b]$: that is, the position $\mathsf{pos}(r)$ of r in the list L of sorted records must lie within $[a, b]$. Equivalently, $\mathsf{rank}(\mathsf{val}(r) - 1) + 1 \geq a$ and $\mathsf{rank}(\mathsf{val}(r)) \leq b$.

[7] We note in passing that this predictability is a necessity for the more standard line of attacks on database encryption schemes based on statistical inference; in this regard our previous attacks are atypical.

Essentially we are performing an approximate reconstruction attack, except instead of outputting an interval of $\mathcal{X}$ containing $\mathsf{val}(r)$, we output an interval of $[1, R]$ containing $\mathsf{pos}(r)$. The point of working with the position of records is that in Step 2 of the algorithm, we will use the auxiliary information about the a priori distribution of values $\mathcal{X}$ to map each position to the most likely associated value.

To see why this makes sense, consider the following minimal example. Assume that the set of values $\mathcal{X}$ is limited to 3 values; and the a priori distribution on values tells us that they occur among records with respective probability 2/3, 1/6, 1/6. Then if we know that the position of r lies within $[1, b]$ for $b \approx R \cdot 2/3$ (which could potentially be learnt from a single query via its rank leakage), we can predict that it is likely to be 1. Compared to the algorithms we have encountered in the previous sections, such an approach can output a guess for the value of a record even when only few range queries are available.

We now explain each step of the algorithm in detail.

Step 1. As noted earlier, determining a possible range for the position of each record is essentially the same as an approximate reconstruction attack in the presence of rank information. Indeed it can be realised in a straightforward manner using the ideas introduced in Section 2.3. Namely, recall that the *partition of records* $\mathcal{P} = \{P(r) : r \in \mathcal{R}\}$ is defined by:

$$P(r) = \Bigg(\bigcap_{\{k : r \in \mathcal{M}_k\}} \mathcal{M}_k \Bigg) \setminus \Bigg(\bigcup_{\{k : r \notin \mathcal{M}_k\}} \mathcal{M}_k \Bigg).$$

In the same way, define the *partition of positions* $\{S(r) : r \in \mathcal{R}\}$ by:

$$S(r) = \Bigg(\bigcap_{\{k : r \in \mathcal{M}_k\}} [a_k + 1, b_k] \Bigg) \setminus \Bigg(\bigcup_{\{k : r \notin \mathcal{M}_k\}} [a_k + 1, b_k] \Bigg).$$

Recall that $[a_k + 1, b_k]$ can be interpreted as the positions of records in $\mathcal{M}_k$: that is, $[a_k + 1, b_k] = \mathsf{pos}(\mathcal{M}_k)$. By looking at the definitions of $P(r)$ and $S(r)$, it is apparent that as a direct result $S(r) = \mathsf{pos}(P(r))$. Thus, $S(r)$ contains precisely the set of positions that record r can occupy in a list of records sorted by value. As result, computing the minimal and maximal possible position of a record r is straightforward: they are precisely $\min(S(r))$ and $\max(S(r))$. This concludes Step 1 of the algorithm. Pseudo-code is provided in Algorithm 4.

Algorithm 4 Computing minimal intervals containing the position of each record.

```
**Input:** query leakage $\mathcal{Q} = (a_k, b_k, \mathcal{M}_k)$ for $k \in [1, Q]$.
**Output:** maps $\mathrm{minPos}, \mathrm{maxPos} : \mathcal{R} \to [1, R]$ s.t. $\forall r, \mathsf{pos}(r) \in [\mathrm{minPos}(r), \mathrm{maxPos}(r)]$.
$S \leftarrow$ empty map
**for all** $r \in \mathcal{R}$ **do**
    $S(r) \leftarrow \left(\bigcap_{\{k : r \in \mathcal{M}_k\}} [a_k + 1, b_k]\right) \setminus \left(\bigcup_{\{k : r \notin \mathcal{M}_k\}} [a_k + 1, b_k]\right)$
    $\mathrm{minPos}(r) \leftarrow \min(S(r))$
    $\mathrm{maxPos}(r) \leftarrow \max(S(r))$
**end for**
**return** $\mathrm{minPos}, \mathrm{maxPos}$
```

Step 2. At this point, for each record r, we have computed a possible range $[a + 1, b]$ for the position of r: we know that record r lies within range $[a + 1, b]$ in the ordered list of records (*i.e.* $\mathsf{pos}(r) \in [a + 1, b]$). From this information, we would like to output an estimate for the value $\mathsf{val}(r)$ of r. For this purpose, we proceed in two steps: first, from the knowledge of a, b, we compute an (approximation of) the distribution of $\mathsf{val}(r)$; second, using this distribution, we output an estimate for $\mathsf{val}(r)$. Let us call these two steps (2a) and (2b); we now explain each step in turn. In the remainder we fix a record r and the corresponding interval of positions $[a + 1, b]$ output by Algorithm 4.

(2a) Note that by construction, a and b are always the rank of some value. Moreover we can attempt to determine these values using knowledge of the auxiliary distribution D: in essence, we evaluate

$\mathsf{rank}(x)$ for all values x based on D, and match a (resp. b) with the closest result. The idea is that this yields a simple model for the distribution of $\mathsf{val}(r)$: namely if $a = \mathsf{rank}(x-1)$ and $b = \mathsf{rank}(y)$, then $\mathsf{val}(r) \in [x, y]$, so we can model the distribution of $\mathsf{val}(r)$ as D restricted to $[x, y]$. We now describe this algorithm in more detail.

The auxiliary distribution D on values tells us that each value $z \in [1, N]$ occurs within records with some probability p_z, with $\sum_{z=1}^{N} p_z = 1$. Let $q_z = \sum_{i=1}^{z} p_i$ denote the cumulative distribution. Note that we no longer assume that every value occurs in at least one record.[8]

For $z \in [1, N]$, $\mathsf{rank}(z)$ can be seen as a random variable whose distribution is determined by D. The distribution of $\mathsf{rank}(z)$ is easily verified to follow a binomial distribution:

$$\Pr[\mathsf{rank}(z) = a] = \binom{R}{a} q_z^a (1 - q_z)^{R-a} \tag{1}$$

and its expected value is $\mathbb{E}(\mathsf{rank}(z)) = Rq_z$.

Given $a \in [1, R]$ and knowing that $a = \mathsf{rank}(z)$ for some z, finding the most likely value of z amounts to choosing z so as to maximise $\Pr[\mathsf{rank}(z) = a]$.[9] Fixing a, observe that the function $q \mapsto q^a(1-q)^{R-a}$ is concave and reaches its maximum for $q = a/R$. Using (1), it follows that the optimal choice of z is either z or $z+1$, for z such that a/R lies within $[q_z, q_{z+1}]$. In other words, as one might expect, if a lies between $\mathbb{E}(\mathsf{rank}(z))$ and $\mathbb{E}(\mathsf{rank}(z+1))$, then the most likely choice for $\mathsf{rank}^{-1}(a)$ is either z or $z+1$. The optimal choice between z and $z+1$ can be determined by computing (1) above and picking the higher of the two values.

In this way, we can compute the most likely values $x, y \in \mathcal{X}$ such that $a = \mathsf{rank}(x-1)$ and $b = \mathsf{rank}(y)$. Assuming that $\mathsf{rank}^{-1}(a)$ and $\mathsf{rank}^{-1}(b)$ are in fact $x-1$ and y, then $\mathsf{val}(r) \in [x, y]$, and we can model the distribution of $\mathsf{val}(r)$ as D restricted to $[x, y]$, *i.e.* each value $t \in [x, y]$ occurs with probability $\left(\sum_{i=x}^{y} p_i\right)^{-1} p_t$. This concludes the first step.

(2b) We now have (an approximation of) the distribution of $\mathsf{val}(r)$. We wish to output an estimate of $\mathsf{val}(r)$. A simple choice is to output its expected value. For the distribution proposed in the previous step, this means outputting as a guess for $\mathsf{val}(r)$ the expectation of a value drawn according to D conditioned on being within $[x, y]$, namely:

$$\left(\sum_{i=x}^{y} p_i\right)^{-1} \left(\sum_{i=x}^{y} i \cdot p_i\right). \tag{2}$$

This concludes the attack.

Variants for Step 2. As a preliminary remark, we stress that although the previous approach for Step 2 is grounded on relevant analysis, and performs well in practice, it remains heuristic. Indeed, define the set of *boundaries* as the image of rank; the name *boundary* comes from the fact that boundaries separate distinct values in the sorted list of records L. From this perspective, each query leaks two boundaries a_k and b_k. At a high level, the problem at hand is to compute the distribution of x, y conditioned on knowing $\mathsf{rank}(x-1) = a$, $\mathsf{rank}(y) = b$, *and* on the knowledge of all other known boundaries a_k and b_k.[10] In the above approach, we disregard boundaries other than a and b, and tackle each of them in isolation, which is a reasonable approximation, but not a perfect one. Although computing

[8] As a side note, asking that values of records follow a distribution D cannot ensure density for $N > 1$, *e.g.* all records could have the same value with non-zero probability. More fundamentally, following a distribution D as above models the values of records as being drawn independently for each record, while asking that every value appears at least once requires some interdependency.

[9] This can be formalised using an application of maximum likelihood estimation.

[10] Empirical solutions for a similar but distinct problem, akin to finding the most likely simultaneous assignment of all boundaries, are proposed in [NKW15, GSB+17].

the likelihood of a given assignment of boundaries to values is simple enough (it follows a multinomial distribution), solving the previous problem that takes into account all boundaries simultaneously, seems to require, at least naively, searching a space of size exponential in N.

A number of trade-offs between accuracy and processing power are possible however. In the remainder, we mention a few such optimisations for step (2a). Regarding step (2b) of the algorithm, we also briefly discuss other choices for the final estimate of $\mathsf{val}(r)$, depending on what metric we wish to optimise.

Starting with step (2a), one possibility is to compute the assignment of both ends of the interval $[a, b]$ simultaneously. That is, instead of computing z_a and z_b to maximise $\Pr[\mathsf{rank}(z_a) = a]$ and $\Pr[\mathsf{rank}(z_b) = b]$ independently, maximise the joint probability $\Pr[\mathsf{rank}(z_a) = a \wedge \mathsf{rank}(z_b) = b]$, which follows a trinomial distribution:

$$\Pr[\mathsf{rank}(z_a) = a \wedge \mathsf{rank}(z_b) = b] = \frac{R!}{a!(b-a)!(R-b)!} q_{z_a}^{a} (q_{z_b} - q_{z_a})^{b-a} (1 - q_{z_b})^{R-b}. \tag{3}$$

Another possibility is to observe that in the original approach as well as the one just above, we first compute the most likely values x, y such that $\mathsf{rank}(x-1) = a$ and $\mathsf{rank}(y) = b$, *then* approximate the distribution of $\mathsf{val}(r)$ by D restricted to $[x, y]$. In other words we are forcing $\mathsf{rank}^{-1}(a)$ and $\mathsf{rank}^{-1}(b)$ to take their most likely values and deducing the distribution of $\mathsf{val}(r)$ from there. We could instead compute the entire distribution of x and y conditioned on $\mathsf{rank}(x-1) = a$ and $\mathsf{rank}(y) = b$, using either (1) or (3); then use this entire distribution to infer that of $\mathsf{val}(r)$. More explicitly, the distribution of $\mathsf{val}(r)$ becomes:

$$\begin{aligned} \Pr[\mathsf{val}(r) = t] = \sum_{(x,y) \in \mathcal{X}^2} & \Pr[\mathsf{val}(r) = t | \mathsf{rank}(x-1) = a \wedge \mathsf{rank}(y) = b] \\ & \cdot \Pr[\mathsf{rank}(x-1) = a \wedge \mathsf{rank}(y) = b] \end{aligned} \tag{4}$$

where the first term is equal to $\Pr[\mathsf{val}(r) = t | \mathsf{val}(r) \in [x, y]] = \left(\sum_{i=x}^{y} p_i\right)^{-1} p_t$ and the second term can be computed using (3). A merit of this approach is that it fully captures the information leaked by $\mathsf{pos}(r) \in [a, b]$. It does, however, remain heuristic as already discussed – in the sense that we are still ignoring the existence of other known boundaries.

In step (2b), we use the approximation D_v of the distribution of $\mathsf{val}(r)$ output by step (2a) to produce an estimate e for $\mathsf{val}(r)$. We chose to output the expected value of x for $x \leftarrow D_v$. If we define the *error* as the difference $x - e$ between $x \leftarrow D_v$ and the estimate e, then the choice of picking the expectation ensures that the mean error is zero, and also minimises the variance of the error.[11] Other metrics we may wish to minimise include the mean of the absolute error $|x - e|$; in that case we should pick the median of the distribution as our estimate; or we may prefer to minimise the median of the absolute error, which amounts to finding an interval $[s, t] \subseteq \mathcal{X}$ of minimal length and probability at least $1/2$, and outputting $(s + t)/2$ (finding such an interval can be done in time $O(N)$).

Finally, we note that the algorithm could output more than simply an estimate of the value. It could, for example, simply output D_v; or, after estimating the most likely values for x and y, output the entire range $[x, y]$. Alternatively, the algorithm could compute a confidence interval around the estimated value from (2). This is straightforward using the approximate distribution output by step (2a). Such a confidence interval would be particularly meaningful if the distribution is computed as in (4), since it would then properly capture situations where a boundary falls in the middle of many successive values with low probability, which can introduce a significant amount of uncertainty.

We note that in our experiments, the simple approach presented in Step 2 already proved to be effective in practice. See further discussion in Section 4.2 below.

Complexity. Step 2 requires a negligible amount of computation (for the variant we chose). Regarding Step 1, as in previous sections, Algorithm 4 can be sped up considerably using the techniques

[11] To see this, if D_v assigns probability p_i to $x = i$, then the variance of the error is $\sum_i p_i (i - e)^2$; it has degree two in e and derivative $2(e - \mathbb{E}[D_v])$, so its minimum is reached for $e = \mathbb{E}[D_v]$.

from Appendix A. In fact Algorithm 5 from Appendix A can be naturally adapted to compute the same output as Algorithm 4: it suffices to replace lines 25 and 27 in Algorithm 5 by minH $\leftarrow \ell_i + 1$ and maxH $\leftarrow \ell_{i+1}$ respectively. In this way Step 1 can be computed very efficiently in $O(Q(N+R))$ operations, in little more time than it takes to read all queries.

4.2 Experimental Results

Since this attack is less amenable to rigorous analysis, we carry out an empirical evaluation by simulating queries on age data. The data we use are from the 2009 Nationwide Inpatient Sample (NIS), from the Healthcare Cost and Utilization Project (HCUP), run by the Agency for Healthcare Research and Quality in the United States [Age09]. (We expect results for any other year would be similar.) The NIS is the largest publicly available inpatient database in the United States that includes discharge data for all types of payers. It includes data at the hospital level – e.g. number of discharges, number of beds – and at the patient discharge level – e.g. demographics, diagnosis, procedure, payer.

We extract the age data (in years) from the patient discharge records of the largest 200 hospitals in the 2009 sample. Each of these largest hospitals has between 13,000 and 122,000 records, approximately, for a total of about 4.9 million records. We simulate range queries on individual hospitals' data and attack the query leakage; the auxiliary distribution we use is obtained by averaging over all 200 largest hospitals' records. Query end points are sampled independently and uniformly at random from $[0, 124]$, the range of valid age values according to the NIS Description of Data Elements. The auxiliary attack does not require the data to be dense and, indeed, some of the ages in this range do not appear in the records of any of the largest 200 hospitals. Further, each hospital's data is not necessarily dense with respect to the auxiliary distribution.

We measure the attack's success by computing the proportion of a hospital's records that was recovered within ϵ, i.e. the fraction of records for which the guessed value is at most ϵN years away from the true value, where N is the number of possible values, 125. It is important to note that ϵ is used only to characterise results and is *not* a parameter of the attack, unlike the approximate reconstruction attack in Section 3. Here, there is no guarantee that the guessed values are within ϵ of the true values.

We display results from our experiments in plots with relative error (ϵ) on the x-axis and cumulative fraction of records on the y-axis. Since ages are in the range $[0, 124]$, the margin of error for all records cannot be higher than 125, which corresponds to $\epsilon = 1$. Perfect reconstruction corresponds to a vertical line at $\epsilon = 0$, while successful attacks have steeply rising curves that reach $y = 1$ for small values of ϵ.

Suitability of the auxiliary distribution. First, we demonstrate the extent to which the performance of this attack is limited by the accuracy of the auxiliary distribution. Although each hospital has over 10,000 records, the per-hospital distributions of ages can vary greatly from the auxiliary distribution. (Such differences could be due to regional demographics or specialised departments, e.g. neonatal, pediatric, or geriatric.) Figure 2 shows the *asymptotic* success of the attack for each of the 200 hospitals: we plot the fraction of records recovered within ϵ as the number of observed queries tends to infinity, meaning that there has been enough leakage to fully determine the partitions of records and the partitions of positions. To measure how closely each hospital's distribution matches the auxiliary distribution, and to investigate the effect of this on the success of our attack, we colour-code each hospital's curve in Figure 2 with the discrete Kolmogorov-Smirnov (K-S) statistic for the per hospital and aggregate distributions. The K-S statistic is a cumulative goodness-of-fit test; for two discrete distributions, it is the maximum absolute difference between their cumulative distribution functions: $KS(q, q') := \max_{z \in \mathcal{X}} |q_z - q'_z|$. The smaller the K-S statistic, the closer the two distributions. If the attack were carried out with exact knowledge of the frequencies, the relative error for all records would be 0 as the number of observed queries tends to infinity. Figure 2 illustrates the importance of the closeness of the auxiliary distribution to the actual distribution: for small K-S values (encoded in dark red), the algorithm generally performs better, recovering more records with a smaller relative error.

In the remainder of this section, we focus on one hospital with over 30,000 records whose distribution's closeness to the auxiliary distribution was about average: its K-S statistic (about 0.098) is near the median (about 0.103).

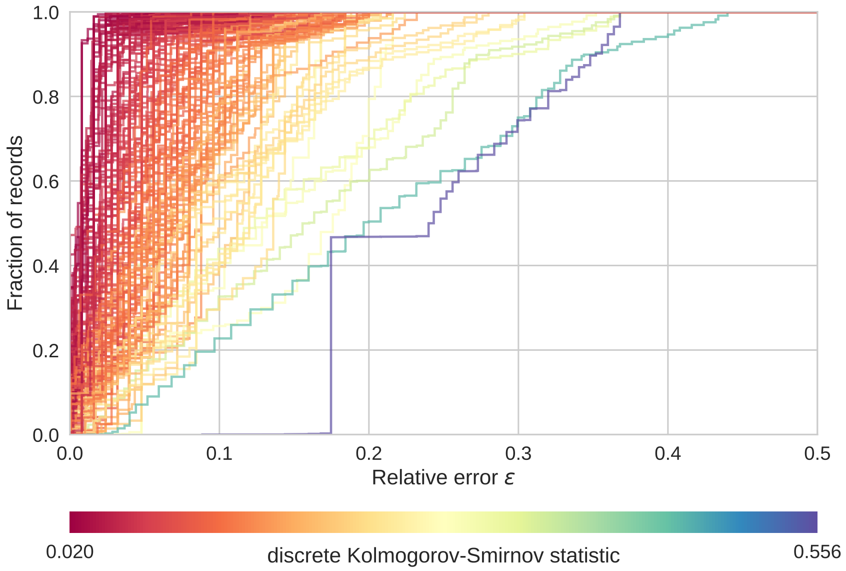
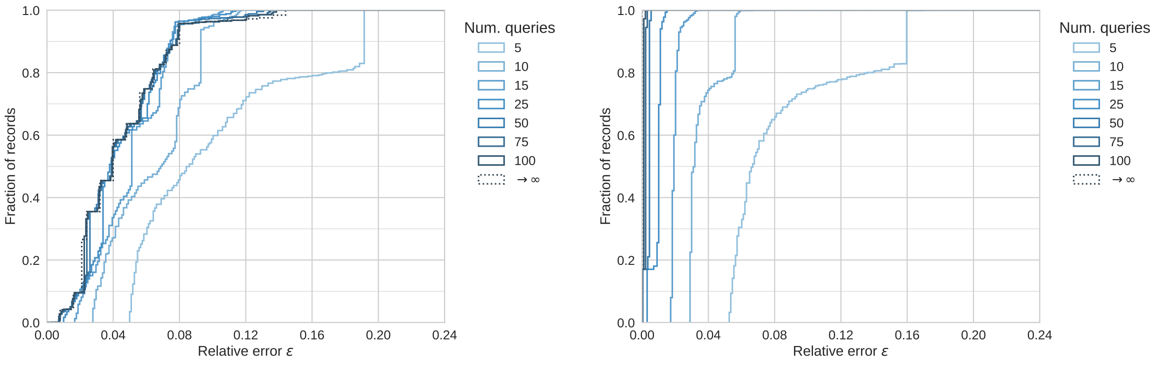


Figure 2: Fraction of records recovered within ϵ of actual age as number of observed queries tends to infinity, for largest 200 hospitals.



(a) Approximate auxiliary information

(b) Exact auxiliary information

Figure 3: Fraction of records recovered within ϵ of actual age for one hospital, averaged over 1000 experiments.

Required number of observed queries. Figure 3a shows the success of the attack on this particular hospital's data, averaged over 1000 experiments, with values assigned to records after 5, 10, 15, 25, 50, 75, and 100 queries. Figure 3b shows what its success would be if the auxiliary information were perfect. Even with this simple heuristic attack and only approximate auxiliary information, the number of queries required to reconstruct most of the database is relatively small. (Recall that for $N = 125$, the number of queries for a full reconstruction attack with rank leakage can be as high as 853.) After observing only 10 queries, an attacker can already guess the ages of 70% of records within 10 years ($\epsilon = 0.08$). After 25 queries, it can guess the ages of 55% of records within 5 years ($\epsilon = 0.04$) After 50 queries, it can guess the ages of 35% of records within 3 years ($\epsilon = 0.024$). After 100 queries, the success of the attack is restricted only by the accuracy of the auxiliary data's distribution.

Relaxing assumptions. Although the auxiliary attack does not assume density of the data, it still makes two important assumptions: that the total number of records is known (which is necessary to obtain point guesses from rank values), and that the set of all record identifiers is known. While we have referred to sets of record identifiers as "access pattern leakage", this information need not come from observing disk accesses at the server – the adversary could learn the record identifiers by intercepting the query result, for instance. In such a setting, the adversary would not be able to learn the set of record identifiers or the total number of records by any methods that require (physical) access to the storage media. We briefly discuss the impact of removing these two assumptions.

- For some distributions – in particular, the age data we used – approximating the total number of records is easy with only a few queries. Consider the following approach: (i) compute the *expected value* of the maximum value in $\mathcal{X}$ that was a query end point in the set of Q queries, then (ii) use the maximum observed rank and the cumulative auxiliary distribution to arrive at an estimate $\tilde{R}$.

  Recall that ranges are chosen independently and uniformly at random from $\mathcal{X} = [1, N]$. Let A_Q denote the maximum value of a query end point after Q queries have been made:

  $$A_Q = \max_{\text{query } k \in \mathcal{Q}} \max \{x_k, y_k\} = \max_{\text{query } k \in \mathcal{Q}} \{y_k\}.$$

  Although this value is unknown, it must correspond to the maximum observed rank and we can compute its expected value. As shown in Appendix B.1, Lemma 1, the probability that the right end point y of a uniformly random range is equal to $z \in \mathcal{X}$ is $\frac{2z}{N(N+1)}$; and so the probability that is less than or equal to z is $\frac{z(z+1)}{N(N+1)}$. Hence the probability that $A_Q = z$ for any $z \in \mathcal{X}$ is:

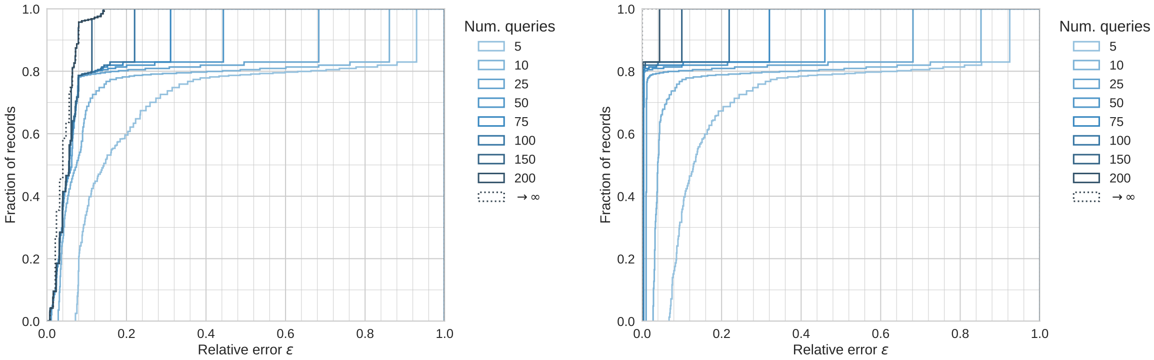
  $$\begin{aligned}\Pr[A_Q = z] &= \Pr[A_Q \le z] - \Pr[A_Q \le z-1] \\ &= \left(\frac{z(z+1)}{N(N+1)}\right)^Q - \left(\frac{(z-1)z}{N(N+1)}\right)^Q.\end{aligned}$$

  Simplifying, we find an expected value:

  $$\mathbb{E}[A_Q] = N - \frac{1}{(N(N+1))^Q} \sum_{z=1}^{N-1} (z(z+1))^Q.$$

  Now, guess that the maximum observed rank, $b_{\max} := \max_{\text{query } k \in \mathcal{Q}} \{b_k\}$, corresponds to the end point $\hat{y}_{max} := \lfloor \mathbb{E}(A_Q) \rceil$. Finally, we can estimate the number of records R using the cumulative distribution function q derived from the auxiliary distribution: $\tilde{R} := \lfloor b_{\max}/q_{\hat{y}_{\max}} \rceil$.

  Returning to our experiment, the expected maximum observed end point is 118.6 after just 10 queries, 122.0 after 25 queries, 123.2 after 50 queries, and 123.6 after 75 queries. This heuristic works well for the age dataset because query end points are sampled uniformly at random, while ages above 110 are infrequent and not dense, so it is unlikely to take more than 10 queries for the maximum rank value (i.e. the number of records) to be observed. For other distributions, perhaps



(a) Approximate auxiliary information

(b) Exact auxiliary information

Figure 4: Fraction of records recovered within ϵ of actual age for one hospital, averaged over 1000 experiments, without assumption that set of record identifiers is known.

the minimum or the mean would be more suitable. Since so few queries are required to estimate the total number of records R given an auxiliary distribution, we are confident that removing this assumption would not have significantly decreased the attack's success in our experiments.

- Next, consider the assumption that the adversary knows the set of all record identifiers. If we remove this assumption, then it is clear that the adversary cannot "reconstruct" any records that have not matched at least one query. Figure 4 shows the results of the experiment when the attacker does not know the set of possible record identifiers before observing any queries, with the aggregate auxiliary distribution (4a) and exact auxiliary information (4b).

  In the case of our experiment, the most significant phenomenon arises from the fact that 17% of our chosen hospital's records have value 0. If no query covers the value 0, then the corresponding 17% of records cannot be reconstructed. This results in a sharp jump depending on whether a query covering the value 0 has been issued, visible in the form of vertical lines at the top of the curves in Figure 4.[12]

  When we assume the set of all record identifiers is known, the attacker recovers all records within an error of about $\epsilon = 0.19$ after seeing only 5 queries and using the approximate auxiliary distribution. Without record identifiers, the IDs of 17% of the records cannot be recovered (much less reconstructed) until an expected number of $2/(N+1) = 63$ queries have been issued. As a result the performance of the attack, visible on Figure 4, is significantly worse than in the case where record identifiers are known. (When comparing these graphs to Figure 3, note that the x-axis now runs from 0 to 1 rather than 0.24 and that the results are for up to 200 queries rather than just 100.) This illustrates the value of knowing the set of record identifiers in our attacks.

5 Conclusions

Building secure databases supporting commonly expected functionality, such as range queries, is a challenging task. Initial solutions based on OPE or ORE offered great usability, but were quickly shown to offer little security in the face of frequency analysis attacks exploiting auxiliary distributions, cf. [NKW15, GSB$^+$17] – and no security at all if the database is dense. Second-generation schemes

[12] Because we average the error over a large number of experiments, the horizontal position of the vertical line at the top of each curve reflects the probability that a query covering the value 0 was issued: indeed if the value 0 has been queried, the corresponding 17% of records are recovered with an error close to 0; while if it was not queried, the records cannot be reconstructed, and are attributed an error of 1.

such as the Lewi-Wu ORE scheme, Arx and FH-OPE seek to increase security while preserving all or most of the functionality of OPE. Indeed, in the case of the Lewi-Wu scheme, Arx, and FH-OPE, significant progress seems to have been achieved against snapshot adversaries: although FH-OPE is still vulnerable to some attacks [GSB+17], the Lewi-Wu scheme as well as Arx offer, at least in principle,[13] a high level of security against snapshot attackers, in a way that precludes the type of statistical inference attacks that worked so powerfully against earlier candidates.

This left open the question of the security offered by such schemes against persistent adversaries, including an honest-but-curious host server (or any adversary having compromised the server for a sufficient length of time). On that front, the generic attack by Kellaris *et al.* [KKNO16] shows that observing the access pattern leakage of $O(N^2 \log N)$ queries is enough to reconstruct the value of all records when the database is dense. Our own attacks reduce the expected number of queries to $N \log(N) + O(N)$ in the same setting. Even for a relatively low number of values $N = 125$, this reduces the required number of queries from around 75,000 to around 800. We also prove a linear upper bound $O(N)$ for approximate reconstruction with a fixed precision (say, 5%). These results come with matching lower bounds, and efficient algorithms (which, in the case of the full reconstruction attack, are also data-optimal).

Furthermore, we investigate the setting where rank information is leaked (as is the case for the Lewi-Wu scheme, Arx, and FH-OPE), and an approximation of the distribution of plaintext values is known to the adversary. In that setting, our experiments on a real-world medical database show that after observing the leakage of as few as 25 queries, the age of a majority of patients could be reconstructed to within 5 years, even with imperfect auxiliary information (with perfect auxiliary information, the precision improves to 2 years, for most records).

All of our results combine to show that, to the best of our knowledge, no known practical scheme supporting range queries achieves a meaningful level of privacy for the client's data with respect to the server hosting the database (or any other persistent adversary).

It would be interesting to analyse to what extent full or approximate reconstruction can be achieved when the density assumption does *not* hold. Such attacks were explored in [KKNO16], where a full reconstruction algorithm requiring the access pattern leakage from $O(N^4 \log N)$ queries was established and shown to be essentially optimal over all data distributions. Still, improvements may well be possible for typical data distributions (as opposed to the pathological ones used to show an $\Omega(N^4)$ lower bound in [KKNO16]). Another interesting extension would be to study the effectiveness of such attacks for non-uniform range queries, for instance using real-world range query samples (although by definition our data-optimal algorithms would still behave optimally, their data requirements may change). On the positive side, an interesting open problem is whether it is possible to build database encryption schemes that satisfy some sensible efficiency criteria (such as being able to locate a record with a given value in logarithmic time, and using $O(1)$ bandwidth from client to server when issuing a query) while offering meaningful security against persistent adversaries.

In the absence of such a scheme, we are not in a position to suggest defences against the attacks presented in this paper, except for employing methods that hide access patterns. This can be done using ORAM techniques, or by splitting the database across multiple servers and using PIR techniques. These approaches are currently rather expensive for use at large scale. They suggest that developing "good enough" oblivious access techniques for stored data, where some security is traded for increased efficiency, would be a fruitful research direction.

Acknowledgements. The authors would like to thank Paul Grubbs and Raluca Ada Popa for their helpful comments on an earlier version of this paper. The first author was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. H2020-MSCA-ITN-2014-643161 ECRYPT-NET. The second and third authors were supported by EPSRC Grant EP/L018543/1.

[13] Counterpoints on the realism of snapshot adversaries are offered in [GRS17].

References

[AEJ+15] Arvind Arasu, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, and Ravi Ramamurthy. Transaction processing on confidential data using Cipherbase. In *2015 IEEE 31st International Conference on Data Engineering*, pages 435–446, April 2015.

[Age09] Agency for Healthcare Research and Quality, Rockville, MD. HCUP Nationwide Inpatient Sample (NIS), Healthcare Cost and Utilization Project (HCUP), 2009. http://www.hcup-us.ahrq.gov/nisoverview.jsp.

[BLR+15] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 563–594. Springer, Heidelberg, April 2015.

[BM97] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 163–192. Springer, Heidelberg, May 1997.

[BPP16] Tobias Boelter, Rishabh Poddar, and Raluca Ada Popa. A secure one-roundtrip index for range queries. Cryptology ePrint Archive, Report 2016/568, 2016. http://eprint.iacr.org/2016/568.

[CDvD+03] Dwaine E. Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G. Edward Suh. Incremental multiset hash functions and their application to memory integrity checking. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 188–207. Springer, Heidelberg, November / December 2003.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 668–679. ACM Press, October 2015.

[CLWW16] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. Practical order-revealing encryption with limited leakage. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 474–493. Springer, Heidelberg, March 2016.

[DDC16] F. Betül Durak, Thomas M. DuBuisson, and David Cash. What else is revealed by order-revealing encryption? In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1155–1166. ACM Press, October 2016.

[FJK+15] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *ESORICS 2015, Part II*, volume 9327 of *LNCS*, pages 123–145. Springer, Heidelberg, September 2015.

[FVY+17] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. Sok: Cryptographically protected database search. In *2017 IEEE Symposium on Security and Privacy*, pages 172–191. IEEE Computer Society Press, May 2017.

[GMN+16] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking web applications built on top of encrypted data. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1353–1364. ACM Press, October 2016.

[GRS17] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. Why your encrypted database is not secure. Cryptology ePrint Archive, Report 2017/468, 2017. http://eprint.iacr.org/2017/468.

[GSB$^+$17] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy*, pages 655–672. IEEE Computer Society Press, May 2017.

[HK16] Florian Hahn and Florian Kerschbaum. Poly-logarithmic range queries on encrypted data with small leakage. In *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*, CCSW '16, pages 23–34, New York, NY, USA, 2016. ACM.

[Ker15] Florian Kerschbaum. Frequency-hiding order-preserving encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 656–667. ACM Press, October 2015.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1329–1340. ACM Press, October 2016.

[Lu12] Yanbin Lu. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *NDSS 2012*. The Internet Society, February 2012.

[LW16] Kevin Lewi and David J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1167–1178. ACM Press, October 2016.

[MSTA17] Jeremy Maitin-Shepard, Mehdi Tibouchi, and Diego F. Aranha. Elliptic curve multiset hash. *The Computer Journal*, 60(4):476–490, 2017.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 644–655. ACM Press, October 2015.

[PBP16] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. Arx: A strongly encrypted database system. Cryptology ePrint Archive, Report 2016/591, 2016. http://eprint.iacr.org/2016/591.

[PKV$^+$14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374. IEEE Computer Society Press, May 2014.

[PW16] David Pouliot and Charles V. Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1341–1352. ACM Press, October 2016.

[RACY16] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. POPE: Partial order preserving encoding. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1131–1142. ACM Press, October 2016.

[YGW+17] Xingliang Yuan, Yu Guo, Xinyu Wang, Cong Wang, Baochun Li, and Xiaohua Jia. Enckv: An encrypted key-value store with rich queries. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 423–435, New York, NY, USA, 2017. ACM.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 707–720. USENIX Association, 2016.

A Fast Reconstruction Algorithms

In this section, we describe fast algorithms for full and approximate reconstruction, with and without rank information. The algorithms presented in Section 2 and 3 are meant to facilitate understanding and analysis: they adhere closely to their design rationale, but leave out major optimisations (although they clearly terminate in polynomial time). By contrast, the algorithms we present in this section are much more efficient; in fact, in the presence of rank information, these new algorithms amount to little overhead over simply *reading* the query leakage.

Chief among the improvements we introduce in this section is a new algorithm for computing the partition of records. While our original algorithms consist mostly of performing numerous set intersections, unions and differences, we show how to realise full and approximate reconstruction attacks in the presence of rank information without explicitly computing *any* such set operation. Our second main improvement targets reconstruction attacks without rank information, and essentially reduces the number of records to the number of values, which in many settings is a major time save; in our experiments from Section 4, values represent the age of patients in a large hospital, ranging up to 120, while the number of records is in the order of $100,000$.

In Section A.1, we describe very efficient full and approximate reconstruction algorithms in the presence of rank information leakage. In Section A.2, we turn to full and approximate reconstruction attacks without rank information leakage, *i.e.* when only the access pattern is leaked.

A.1 Fast Reconstruction with Rank Information

Sequence hashing. Given some hash function H with fixed-length output, we define the *sequence hash* H_S of the ordered sequence of integers $k_1 < \cdots < k_n$ inductively by $H(\varnothing) = 0$ and:

$$H_S(\langle k_1, \ldots, k_n \rangle) = H(H(\langle k_1, \ldots, k_{n-1} \rangle)||k_n)$$

where $||$ denotes the concatenation of bit strings. It is easy to see that the collision resistance of H_S reduces to the collision resistance of the underlying hash function H.

We note that the above technique is a rudimentary form of set hashing, which exploits the fact that its inputs are ordered. In some sense, what we want in the algorithm below is actually a set hash, but because its inputs happen to naturally occur in a fixed order, we can forgo set hashing techniques in favour of the simple sequence hashing technique above (which is cheaper and comes with a reduction to the underlying hash function that is both more immediate and tighter than set hashing techniques). For more information on set hashing, multiset hashing and incremental hashing, we refer the interested reader to [BM97, CDvD+03, MSTA17].

The full reconstruction algorithm. Assume the adversary has access to Q queries on ranges $[x_k, y_k]$, $k \leq Q$. We begin with the full reconstruction attack in the presence of rank information, described in Algorithm 1. In this setting, the leakage of queries observed by the adversary is $\mathcal{Q} = \{(a_k, b_k, \mathcal{M}_k)\}$, with $a_k = \mathsf{rank}(x_k - 1)$, $b_k = \mathsf{rank}(y_k)$, $\mathcal{M}_k = \mathcal{S}_{[x_k, y_k]}$.

The first step of the full reconstruction algorithm essentially amounts to computing the *partition of records* $\mathcal{P}$ (cf. Section 2.3). The partition of records $\mathcal{P}$ is defined by $\mathcal{P} = \{P(r) : r \in \mathcal{R}\}$, with:

$$P(r) = \left(\bigcap_{\{k : r \in \mathcal{M}_k\}} \mathcal{M}_k \right) \Big\backslash \left(\bigcup_{\{k : r \notin \mathcal{M}_k\}} \mathcal{M}_k \right).$$

We now propose an alternate approach to computing all $P(r)$'s simultaneously and efficiently. The key observation is that $P(r)$ is entirely determined by $\{k : r \in \mathcal{M}_k\}$. With this in mind, we proceed as follows. Create a table T mapping the set of records $\mathcal{R}$ to sets of integers. At first all sets in the image of T are empty. The algorithm processes each query in turn. When leakage $(a_k, b_k, \mathcal{M}_k)$ is received, for every $r \in \mathcal{M}_k$, the query handle k is added to $T(r)$. Once all queries are processed, observe that $T(r) = \{k : r \in \mathcal{M}_k\}$. Hence $P(r) = \{r' : T(r') = T(r)\}$: the partition of records $\mathcal{P}$ is exactly the partition induced by the preimages of T (*i.e.* the set of equivalences classes for the relation $r \equiv r'$ iff $T(r) = T(r')$).

Thus, we have computed $\mathcal{P}$ (or an equivalent representation of $\mathcal{P}$, which will actually prove more useful) at minimal cost: essentially, one random memory access per record matched by each query. However two problems remain: $O(QN)$ memory is required, which may be rather large, and testing $T(r) = T(r')$ means testing the equality of two sets of of size $O(Q)$. Fortunately, both issues can be negated by using sequence hashing: instead of having $T(r) = \{k : r \in \mathcal{M}_k\}$, we build T so that:

$$T(r) = H_S(\langle k : r \in \mathcal{M}_k \rangle)$$

where H_S is the sequence hash function introduced earlier in this section.

It is clear that such a T can still be computed as before by processing queries on the fly (when processing each query $q_k = \mathcal{M}_k$ in order, for each $r \in \mathcal{M}_k$, $T(r)$ is replaced by $H(T(r)||k)$), while consuming only $O(N)$ memory and, barring some collision of negligible probability in the sequence hash, allowing equality tests in constant time. The price to pay is a negligible probability of failure, as we require that there should be no collision between the sequence hashes of (up to) N distinct sequences of length at most Q (which directly reduces to there being no collision between at most QN hash values for the underlying hash function H). In the remainder we assume that no such collision occurs. This concludes the *partitioning* step of the algorithm.

It remains to sort the elements of $\mathcal{P}$ by value (recall that $|\mathcal{P}| = N$, which is required for full reconstruction, implies that every element of $\mathcal{P}$ corresponds to a single value). The key point here is that we can build a second table L in exactly the same way that T was built, except that the role of $\mathcal{M}_k$ is played by $[a_k + 1, b_k]$.[14] To be explicit: L is a table mapping $[1, R]$ to a sequence of hash outputs, initialised at 0. When processing the k-th query, $L(i)$ is replaced by $H(L(i)||k)$ for every $i \in [a_k + 1, b_k]$.

The crux of the matter is as follows: recall that $[a_k + 1, b_k]$ is, at least conceptually (and even concretely, in the case of Arx-Range), the positions of records matched by the query in a list of records ordered by value. As in Section 4.1, let us name pos the mapping from records $\mathcal{R}$ to their position in this ordered list. Then as already observed in that section, $\mathsf{pos}(\mathcal{M}_k) = [a_k + 1, b_k]$: $\mathcal{M}_k$ and $[a_k + 1, b_k]$ contain the same records, but represented respectively by their IDs and by their positions in the ordered list of records. Since L is built exactly like T, except $[a_k + 1, b_k]$ is used in place of $\mathcal{M}_k$, we end up with a mapping L such that $L = T \circ \mathsf{pos}$.

In the end, for every record r, the positions i such that $L(i) = T(r)$ reveal exactly the possible positions of r among the list of ordered records. In particular, if $|\mathcal{P}| = N$, then the number of distinct values occurring in L before the first occurrence of $T(r)$ directly reveals the value of the record. This concludes the description of the algorithm. Pseudo-code is postponed until the approximate reconstruction attack, which is essentially identical, as discussed below.

While computing L costs no more than computing T, if $N \ll R$ (which is often expected to be the case), L can be computed much more efficiently. Indeed, at all steps of the algorithm, L (viewed

[14] In the terminology of Section 4.1, L represents the *partition of positions* in the same sense that T represents the partition of records.

as a list of hash values rather than as a mapping) is partitioned into at most N distinct intervals. As a result, L can be more compactly represented (and updated), as a partition of intervals, rather than a mapping. More precisely, L is represented as an ordered list of end points, and to each interval between an end point and the next is associated a hash value. This brings down the memory cost of representing L from $O(R)$ to $O(N)$; and likewise for the time complexity needed to update L per query.

The approximate reconstruction algorithm. The full reconstruction algorithm presented just above applies almost as is to approximate reconstruction. Indeed, as we have seen, the mapping L directly reveals the possible positions of each record r in the ordered list of records (via the positions of $T(r)$ in L viewed as a list). Hence the number of distinct values before the first occurrence of $T(r)$ in L (viewed as a list), and the number of distinct values after the last occurrence of $T(r)$ in L, determine respectively the lowest possible value and the highest possible value of record r. Pseudo-code is provided in Algorithm 5 (including the optimisation that L is represented via its interval partition, as discussed above).[15] Although we do not prove it, it is also clear that the algorithm is data-optimal.

Complexity of Algorithm 5. Computing T and L costs $O(Q(N+R))$ operations (loop on line 3). Computing the mapping from sequence hashes to values (lines 22-28) costs $O(N)$ operations, and finally mapping records to value ranges (lines 29-32) costs $O(R)$ operations. Concretely, assuming $N \ll R$, the dominant cost is the computation of T and L, which mainly costs two memory accesses and hash computations per record matched in each query (less if the representation of L is optimised, as in Algorithm 5), so full and approximate reconstruction cost remarkably little overhead beyond simply reading the query leakage.

A.2 Fast Reconstruction without Rank Information

We now turn to full and approximate reconstruction attacks without rank information, *i.e.* with only access pattern leakage (Algorithms 2 and 3). The general outline of our approach is as follows. For both algorithms, we begin with a partitioning step to find the partition of records $\mathcal{P}$, as in the previous section. We then perform a new *reduction* step that amounts to reducing the number of records from R to N. Once the partitioning and reduction step are completed, the remainder of Algorithms 2 and 3 is executed without much alteration.

While the partitioning and sequence hashing techniques from the previous section provide a major speed boost for virtually all parameter ranges, it should be noted that the reduction step introduced in this section is only meaningful if $N \ll R$. We now go through each step in more detail.

Partitioning step. The partition of records $\mathcal{P}$ is computed as in the previous section. The complexity of this step is $O(Q(N+R))$.

Reduction step. We call *points* the elements of $\mathcal{P}$. Note that every query is a union of points. More generally, throughout all the algorithms we consider, points are the finest granularity at which records are accessed. Thus, insofar as $N \ll R$, it makes sense to replace all sets of records by sets of points. This is the idea of the reduction step. Concretely, points can be represented by their sequence hash, and the map T from Section A.1 provides a mapping from actual record IDs to points.

In the *reduction* step, we replace the set of records by $\mathcal{P}$ (so that there are now at most N records), and, accordingly, replace every query q (assimilated to its set of matched records $\mathcal{M}$) by the corresponding set of points, which is precisely $T(q)$. This requires computing T (one table lookup) for every matched record in a query. Hence the time complexity of this step is again $O(Q(N+R))$.

[15] Algorithm 5 determines the minimal interval of values that a record can be inferred to belong to; technically, it does not check whether a given precision ϵN is achieved. To do so, it suffices to check at the end of the algorithm that $\forall r, \mathrm{maxVal}(r) - \mathrm{minVal}(r) \leq \epsilon N$.

Algorithm 5 Fast approximate reconstruction attack with rank information.

```
FAST-RECONSTRUCTION($\mathcal{Q}$):
Input: set of queries $\mathcal{Q} = \{(a_k, b_k, \mathcal{M}_k)\}$, hash function $H$ into $\{0,1\}^h$.
Output: maps minVal, maxVal : $\mathcal{R} \to \mathcal{X}$ s.t. $\forall r, \mathsf{val}(r) \in [\text{minVal}(r), \text{maxVal}(r)]$.
$T \leftarrow$ map $\mathcal{R} \to \{0,1\}^h$ initialised at 0
$L \leftarrow$ list of elements in $[0, R] \times \{0,1\}^h$, initialised at $\langle(0,0), (R,0)\rangle$
for all $(a_k, b_k, \mathcal{M}_k) \in \mathcal{Q}$ do                          ▷ Partitioning step
    for all $r \in \mathcal{M}_k$ do
        $T(r) = H(T(r)||k)$
    end for
    $\langle(\ell_1, h_1), \ldots, (\ell_{|L|}, h_{|L|})\rangle \leftarrow L$
    Find $i$ s.t. $\ell_i \le a_k < \ell_{i+1}$
    if $\ell_i = a_k$ then
        $(\ell_i, h_i) \leftarrow (\ell_i, H(h_i||k))$
    else
        Insert $(a_k, H(h_i||k))$ before $(\ell_{i+1}, h_{i+1})$
    end if
    while $\ell_{i+1} < b_k$ do
        $i \leftarrow i + 1$
        $(\ell_i, h_i) \leftarrow (\ell_i, H(h_i||k)$
    end while
    if $\ell_{i+1} \neq b_k$ then
        Insert $(b_k, h_i)$ before $(\ell_{i+1}, h_{i+1})$
    end if
end for
minH, maxH, minVal, maxVal $\leftarrow$ empty maps                            ▷ Ranking step
for all $1 \le i < |L|$ do
    if minH($h_i$) undefined then
        minH($h_i$) $\leftarrow i$
    end if
    maxH($h_i$) $\leftarrow N - |L| + i$
end for
for all $r \in \mathcal{R}$ do
    minVal($r$) $\leftarrow$ minH($T(r)$)
    maxVal($r$) $\leftarrow$ maxH($T(r)$)
end for
return minVal, maxVal
```

Full reconstruction without rank information. The partitioning step covers lines 2-5 of Algorithm 2. After ensuring $|\mathcal{P}| = N$, we can perform the reduction step. The rest of the algorithm is executed as is, with points in place of records (so $R = N$). This involves two loops (lines 10 and 17) on N elements; within each loop, the most expensive operations scan all queries and perform some set operations on them, resulting in time complexity $O(QN)$ for each iteration. Overall this step costs $O(QN^2)$ operations.

Approximate reconstruction without rank information. The partitioning and reduction steps are precomputed, before Algorithm 3 is run, so that $R = N$. There is a single main loop on N elements; and as before, it is straightforward to see that all steps within an iteration of the loop cost $O(QN)$ operations, with the sole exception of line 3. Indeed, on line 3, a naive implementation would cost $O(Q^2N)$ operations.

Instead, we propose an alternate approach to finding q_L and q_R on line 3 that only requires $O(QN)$

operations. Let $\mathcal{Q}$ denote the set of queries, where each query is identified with the set of matched records. Let $f(r) = \{q \in \mathcal{Q} : M \subseteq q, r \in q\}$, where $M := \bigcap_{\{k : r \in \mathcal{M}_k\}} \mathcal{M}_k$ is defined on line 2 of the algorithm (note that M depends on r).

For every record r, compute $|f(r)|$ (by setting up an array of counters, one for each r, and scanning once through every query, this costs $O(QN)$ operations), and pick a record $r_L \notin M$ maximising this value.[16] Then pick q_R of maximal size among queries such that $M \subseteq q_R$ and $r_L \notin q_R$. Then pick q_L of maximal size among queries such that $M \subseteq q_L$ and $q_L \cap q_R = M$. The time complexity of this algorithm is clearly $O(QN)$ operations. We claim that it outputs the correct sets q_L and q_R expected in line 3 of Algorithm 3.

To see this, switch the perspective from sets of records to the corresponding sets of values. By construction M is an intersection of queries, so it covers an interval of values $[x_M, y_M]$. Without loss of generality $x_L = \mathsf{val}(r_L) < x_M$. The crucial point is that by construction of r_L, any query $q \supset M$ containing a record r with $x_L \leq \mathsf{val}(r) < x_M$ must also contain r_L: otherwise r would directly witness the fact that r_L fails to maximise $|f(r)|$ (indeed, by nature of intervals, any query containing M and r_L must contain r, so $|f(r)| \geq |f(r_L)|$; but q belongs to $f(r)$ and not $f(r_L)$ so the inequality is strict). Hence q_R is the largest query containing M and *not* covering any value $< x_M$; likewise q_L is the largest query containing M and *not* covering any value $> y_M$; and we are done.

Overall complexity. In both cases the algorithms themselves cost $O(QN^2)$ operations, after the partitioning and reduction steps, which cost $O(Q(N+R))$ operations. Hence the final tally is $O(Q(N^2+R))$ operations.

B Upper Bounds on Data Complexity

In this section, we prove upper bounds on the expected number of queries necessary for Algorithms 1, 2, and 3 to succeed. In each case a lower bound on the success probability of the algorithm (as a function of the number of queries) is also provided. We begin with two useful lemmas.

B.1 Basic Distribution Lemmas

Recall that $\mathcal{U}$ denotes the uniform distribution on non-empty intervals in $\mathcal{X} = [1, N]$.

Lemma 1. *For all $k \in [1, N]$:*

$$\Pr\left[x = k : [x, y] \leftarrow \mathcal{U}\right] = \frac{2(N+1-k)}{N(N+1)}$$
$$\Pr\left[y = k : [x, y] \leftarrow \mathcal{U}\right] = \frac{2k}{N(N+1)}.$$

Proof. Since we allow singleton ranges, but not empty ones, there are $N(N+1)/2$ choices for $[x, y]$. Moreover for any $k \in [1, N]$, exactly $N + 1 - k$ of those choices satisfy $x = k$ (one for each choice of $y \geq k$). Hence the distribution of x for $[x, y] \leftarrow \mathcal{U}$ is as claimed. By the reflection symmetry of the problem (flipping $[1, N]$, replacing k by $N + 1 - k$) we get the second equality. □

Lemma 2. *Assume N is a multiple of 4. Suppose we have drawn Q ranges $r = [x, y] \leftarrow \mathcal{U}$. Let E^L_{half} denote the event that there exist two ranges r, r' such that $r \cup r' = [1, N/2 + 1]$ and $r \cap r' \neq \varnothing$. Let E^R_{half} denote the event that there exist two ranges r, r' such that $r \cup r' = [N/2 + 1, N]$ and $r \cap r' \neq \varnothing$.*[17] *Then:*

$$\Pr\left[E^L_{half}\right] \geq \Pr\left[E^R_{half}\right] \geq 1 - 2e^{-Q/(2N+2)}.$$

[16] The case $M = \mathcal{R}$ can be safely ignored, since it can only lead to a trivial approximate reconstruction.

[17] There is a small asymmetry here: $[1, N/2+1]$ covers $N/2+1$ values while $[N/2+1, N]$ covers $N/2$ values. This will be helpful in the proof of Proposition 3.

Proof. Let us begin with E^R_{half}. Let E^R_1 denote the event that we have drawn a range $r = [N/2+1, y]$ for some $y \in [3N/4+1, N]$. Let E^R_2 denote the event that we have drawn a range $r' = [x, N]$ for some $x \in [N/2+1, 3N/4]$. If both E^R_1 and E^R_2 occur, then E^R_{half} clearly occurs.

There are $N(N+1)/2$ possible ranges, among which we draw uniformly at random. In the definition of range r (resp. r') for event E^R_1 (resp. E^R_2), there are $N/4$ choices for the value of x (resp. y). It follows that for a single draw:

$$\Pr\left[E^R_1\right] = \Pr\left[E^R_2\right] = \frac{N/4}{N(N+1)/2} = \frac{1}{2(N+1)}$$

Hence after Q independent draws:

$$\begin{aligned}\Pr\left[\neg E^R_1\right] = \Pr\left[\neg E^R_2\right] &= \left(1 - \frac{1}{2(N+1)}\right)^Q \\ &= e^{Q\ln\left(1-\frac{1}{2(N+1)}\right)} \\ &\leq e^{-\frac{Q}{2(N+1)}}\end{aligned}$$

Hence we have:

$$\begin{aligned}\Pr\left[E^R_{half}\right] &\geq \Pr\left[E^R_1 \cap E^R_2\right] \\ &\geq 1 - \Pr\left[\neg E^R_1\right] - \Pr\left[\neg E^R_2\right] \\ &\geq 1 - 2e^{-Q/(2N+2)}.\end{aligned}$$

We now show $\Pr\left[E^L_{half}\right] \geq \Pr\left[E^R_{half}\right]$. By symmetry, the probability of the event E': there exist r, r' such that $r \cup r' = [1, N/2]$ and $r \cap r' \neq \varnothing$, is equal to E^R_{half}. Thus it suffices to show that this event is less likely than E^L_{half}. To see this, let f denote the permutation on ranges in $[1, N]$ defined by:

$$f([x,y]) = \begin{cases} [x, N/2+1] & \text{if } y = N/2; \\ [x, N/2] & \text{if } y = N/2+1 \text{ and } x \leq N/2; \\ [x,y] & \text{otherwise.} \end{cases}$$

The point of f is that it maps any pair of ranges r, r' witnessing E' into a pair witnessing E^L_{half} (the converse is not true, as shown by the pair $[1, N/2+1], [N/2+1, N/2+1]$). As a consequence, if we consider sequences of Q queries, then if some sequence $\mathcal{Q}$ satisfies E' we have that $f(\mathcal{Q})$ satisfies E^L_{half}. Since f is a permutation, this shows that the number of sequences of queries satisfying E^L_{half} is higher than the number of sequences of queries satisfying E'. Because we are working with the uniform distribution, this implies:

$$\Pr\left[E^L_{half}\right] \geq \Pr\left[E'\right] = \Pr\left[E^R_{half}\right]. \qquad \square$$

B.2 Upper Bound for the Full Reconstruction Attack with Rank

Proposition 1. *Assume N is a multiple of 4. Assume that the database is dense, and range queries are drawn uniformly at random. Then the probability of success of Algorithm 1 after Q queries is lower-bounded by:*

$$1 - 2e^{-Q/(2N+2)} - Ne^{-Q/N}.$$

Moreover, the expected number of queries before the algorithm succeeds is upper-bounded by:

$$N\log(N) + O(N).$$

Concretely, for $N \geq 27$, it is upper-bounded by $N(\log(N)+1) + 4.4\sqrt{N} + 4 \leq N(\log(N)+2)$.

Proof. Let $\mathcal{Q} = \{q_k = (a_k, b_k, \mathcal{M}_k) : k \leq Q\}$ denote the set of queries. For each $k \leq Q$, let $[x_k, y_k] \subseteq [1, N]$ denote the range of values on which the k-th query q_k bears, *i.e.* $\mathcal{M}_k = \{r \in \mathcal{R} : \mathsf{val}(r) \in [x_k, y_k]\}$. From the discussion in Section 2.3, recall that Algorithm 1 succeeds iff the *partition of values* $\mathcal{V}$ contains N elements. It is easy to see that this is equivalent to the condition that for all $v \in [1, N]$, $I(v) = \{v\}$, where $I(v)$ is defined by:

$$I(v) = \bigcap_{v \in [x_k, y_k]} [x_k, y_k] \setminus \left(\bigcup_{v \notin [x_k, y_k]} [x_k, y_k] \right). \tag{5}$$

In other words, Algorithm 1 succeeds iff the following event occurs:

$$E = [\forall v \in [1, N], \{v\} = I(v)].$$

Our goal in this proof is to lower-bound the probability of E as a function of the number Q of queries. For this purpose, we split E into a a set of simpler events whose joint occurrence implies E.

First, we define event E^R_{half}: there exist two queries $q_k, q_{k'}$ such that $[x_k, y_k] \cup [x_{k'}, y_{k'}] = [N/2+1, N]$. Lemma 2 provides a lower bound for the probability of this event.[18] In addition, we define events E^L_x: there exists a query with left end point x; and E^R_y: there exists a query with right end point y.

By analogy to the coupon collector's problem, we call E^L_x a *left* coupon, and E^R_y a *right* coupon. The idea is that if we collect all left coupons in $[1, N/2]$ and all right coupons in $[N/2+1, N]$, in addition to E^R_{half}, then E happens and the algorithm succeeds. This is expressed by the following claim.

Claim 1.

$$E \supseteq E^R_{\mathit{half}} \cap \left(\bigcap_{x=1}^{N/2} E^L_x \right) \cap \left(\bigcap_{y=N/2+1}^{N} E^R_y \right).$$

Proof. Pick $v \in [1, N]$. Assume $v \leq N/2$. The proof for $v > N/2$ is similar. From the definition of $I(v)$, it is clear that $v \in I(v)$. Conversely, let $w \in [1, N] \setminus \{v\}$. We want to show $w \notin I(v)$. We distinguish three cases.

1. If $w < v$, E^L_v implies $w \notin I(v)$ as w cannot belong to the intersection on the left side of (5).
2. If $w \in [v+1, N/2]$, E^L_w implies $w \notin I(v)$ as w must belong to the union on the right side of (5).
3. If $w \in [N/2+1, N]$, E^R_{half} implies $w \notin I(v)$ as w must belong to the union on the right side of (5).

A small subtlety worth pointing out is that for the case $v > N/2$, the third case becomes $w \in [1, N/2]$, and E^R_{half} is sufficient to handle it as it implies that the intersection on the left side of (5) must be contained in $[N/2+1, N]$. As a result the reflection E^L_{half} of E^R_{half} is not needed in this proof. □

It now remains to lower-bound the probability of $\bigcap_{x=1}^{N/2} E^L_x$ and $\bigcap_{y=N/2+1}^{N} E^R_y$. Note that these two events are completely symmetric via the reflection that flips $[1, N]$, replacing k by $N+1-k$ and switching the roles of left and right coupons. In particular the two aforementioned events must have

[18] There are two small abuses of notation: first, in Lemma 2, E^R_{half} was an event on ranges, whereas here E^R_{half} is an event on queries; in this case we assimilate a query with the corresponding queried range. Second, the event E^R_{half} as defined in Lemma 2 also requires $[x_k, y_k] \cap [x_{k'}, y_{k'}] \neq \varnothing$, so it is a strict subset of E^R_{half} as defined here, but since it is used as a lower bound this works to our advantage.

the same probability. Hence we can restrict our attention to the first event:

$$\begin{aligned}
\Pr\left[\bigcap_{x=1}^{N/2} E_x^L\right] &\geq 1 - \sum_{x=1}^{N/2} \Pr\left[\neg E_x^L\right] \\
&= 1 - \sum_{x=1}^{N/2} \left(1 - \frac{2(N+1-x)}{N(N+1)}\right)^Q && \text{by Lemma 1} \\
&\geq 1 - \frac{N}{2}\left(1 - \frac{2(N/2+1)}{N(N+1)}\right)^Q \\
&\geq 1 - \frac{N}{2}\left(1 - \frac{1}{N}\right)^Q \\
&\geq 1 - \frac{N}{2} e^{-Q/N}.
\end{aligned}$$

Hence we get:

$$\begin{aligned}
\Pr[E] &\geq 1 - \Pr\left[\neg E_{half}^R\right] - 2\Pr\left[\neg \bigcap_{x=1}^{N/2} E_x^L\right] && \text{by Claim 1} \\
&\geq 1 - 2e^{-Q/(2N+2)} - Ne^{-Q/N}. && \text{by Lemma 2}
\end{aligned}$$

This concludes the proof of the first statement.

Let T be the random variable denoting the first time t such that the event E occurs given uniformly distributed range queries $q_1, \ldots, q_t$. What we have proven so far is $\Pr[T > Q] \leq 2e^{-Q/(2N+2)} + Ne^{-Q/N}$. It remains to compute $\mathbb{E}[T]$.

$$\begin{aligned}
\mathbb{E}[T] &= \sum_{t=1}^{\infty} t \Pr[T = t] \\
&= \sum_{t=0}^{\infty} \Pr[T > t] \\
&\leq N \log N + 1 + \sum_{t=\lceil N \log N \rceil}^{\infty} \left(2e^{-t/(2N+2)} + Ne^{-t/N}\right). \qquad (6)
\end{aligned}$$

Let us focus on the two sub-terms appearing in the sum:

$$\begin{aligned}
\sum_{t=\lceil N \log N \rceil}^{\infty} e^{-t/(2N+2)} &\leq e^{-\frac{N\log(N)}{2N+2}} \sum_{t=0}^{\infty} e^{-t/(2N+2)} \\
&= \frac{N^{-1/2} e^{\log(N)/(2N+2)}}{1 - e^{-1/(2N+2)}} \\
&\leq 1.1 N^{-1/2}(2N+3) && \text{for } N \geq 13 \\
&\leq 2.2\sqrt{N} + 1 && \text{for } N \geq 13 \\
N \sum_{t=\lceil N \log N \rceil}^{\infty} e^{-t/N} &\leq \sum_{t=0}^{\infty} e^{-t/N} \\
&= \frac{1}{1 - e^{-1/N}} \\
&\leq N + 1.
\end{aligned}$$

where lines 3 and 7 use the fact that $1/(1-e^{-1/x}) \leq 1+x$ for $x \geq 0$.

Reinjecting into (6), we conclude that for $N \geq 13$:

$$\mathbb{E}[T] \leq N\log(N) + N + 4.4\sqrt{N} + 4.$$

For $N \geq 27$, $4.4\sqrt{N} + 4 \leq N$ so in that case we have:

$$\mathbb{E}[T] \leq N(\log(N) + 2). \qquad \square$$

B.3 Upper Bound for the Full Reconstruction Attack Without Rank

Proposition 2. *Assume N is a multiple of 4. Assume that the database is dense, and range queries are drawn uniformly at random. Then the probability of success of Algorithm 2 after Q queries is lower-bounded by:*

$$1 - 4e^{-Q/(2N+2)} - Ne^{-Q/(N+1)}.$$

Moreover, the expected number of queries before the algorithm succeeds is upper-bounded by:

$$N\log(N) + O(N).$$

Concretely, for $N \geq 26$, it is upper-bounded by $(N+1)(\log(N)+1) + 8\sqrt{N} + 6 \leq N(\log(N)+3)$.

Proof. Let $\mathcal{Q} = \{q_k = (\mathcal{M}_k) : k \leq Q\}$ denote the set of queries. For each $k \leq Q$, let $[x_k, y_k] \subseteq [1, N]$ denote the range of values on which the k-th query q_k bears, *i.e.* $\mathcal{M}_k = \{r \in \mathcal{R} : \mathsf{val}(r) \in [x_k, y_k]\}$.

The proof closely follows the case where rank information is available. Let E denote the event that Algorithm 2 succeeds. We are going to define several events whose joint occurrence implies E. In fact, these events will be almost the same as those of Proposition 1. Then in a second step we will lower-bound the probability of these events.

Define event E^R_{half}: there exist two queries q_k, $q_{k'}$ such that $[x_k, y_k] \cup [x_{k'}, y_{k'}] = [N/2+1, N]$ and $[x_k, y_k] \cap [x_{k'}, y_{k'}] \neq \varnothing$. Likewise, define event E^L_{half}: there exist two queries q_k, $q_{k'}$ such that $[x_k, y_k] \cup [x_{k'}, y_{k'}] = [1, N/2]$ and $[x_k, y_k] \cap [x_{k'}, y_{k'}] \neq \varnothing$. Next, we define events E'^L_x: there exists a query q_k with left end point $x_k = x$ and length at least 2 (*i.e.* $y_k - x_k + 1 \geq 2$). Likewise, let E'^R_y denote the event: there exists a query with right end point y and length at least 2 (in the same sense). Thus the definition of E^R_{half} matches that of Lemma 2, and E^L_{half} is its reflection; while E'^L_x and E'^R_y are identical to E^L_x and E^R_y from Proposition 1, except for the additional requirement that queries should have length at least 2.

We call E'^L_x a *left* coupon, and E'^R_y a *right* coupon. The idea is that if we collect all left coupons in $[1, N/2]$ and all right coupons in $[N/2+1, N]$, in addition to E^L_{half} and E^R_{half}, then E happens and the algorithm succeeds. This is expressed by the following claim (which is of course the counterpart of Claim 1).

Claim 2.

$$E \supseteq E^L_{half} \cap E^R_{half} \cap \left(\bigcap_{x=1}^{N/2} E'^L_x\right) \cap \left(\bigcap_{y=N/2+1}^{N} E'^R_y\right).$$

Proof. Throughout, we assume that events E^L_{half}, E^R_{half}, $\bigcap_{x=1}^{N/2} E^L_x$ and $\bigcap_{y=N/2+1}^{N} E^R_y$ occur. Our goal is to show that E happens, *i.e.* Algorithm 2 succeeds. There are three lines where Algorithm 2 can fail: lines 7, 14, and 24. We examine each of them in turn.

First, notice that the first fail condition on line 7 in Algorithm 2 is exactly equivalent to the fail condition on line 7 in Algorithm 1. Moreover the events E'^L_x, E'^R_y are strict subsets of their counterparts E^L_x, E^R_y in Claim 1 from the proof of Proposition 1 (and so is E^R_{half}). By Claim 1, it follows that Algorithm 1 cannot fail, and hence Algorithm 2 also cannot fail at line 7.

We now turn our attention to line 14. At this step the set of records has successfully been partitioned into N subsets, contained in partition $\mathcal{P}$. Each element of $\mathcal{P}$ is the set of records corresponding to a particular value in $[1, N]$. We call the elements of $\mathcal{P}$ points. The remainder of the algorithm is aiming to sort the set $\mathcal{P}$ of points according to their value. To start with, it is attempting to find an extremal point. To do so, it starts with an arbitrary query corresponding to some range $[x, y] \subsetneq [1, N]$, and progressively extends it using the following process.

At each iteration the variable S holds a set of records corresponding to some interval $[x, y] \subsetneq [1, N]$ of values. Then it looks for a query q overlapping S and such that $S \cup q$ is not the full set $\mathcal{R}$, and adds q to S. Iterations continue until no such q exists. The fail condition on line 14 is triggered iff at the end of this process, $\mathcal{R} \setminus S$ is not reduced to a single point. We now show that this is impossible.

Let $[x, y]$ denote the interval of values corresponding to S. By construction there cannot exist q such that q overlaps S and $q \cup S \neq \mathcal{R}$. First, we show that $x > 2$ is impossible. Indeed in that case, if $x \leq N/2 + 1$ (resp. $x > N/2 + 1$) then E'^{L}_{x-1} (resp. E^{R}_{half}) would yield a suitable q and hence a contradiction. So $x \in [1, 2]$. Similarly it must hold that $y \in [N - 1, N]$. Moreover by construction of S, $[x, y] = [1, N]$ is impossible, and $[x, y] = [2, N - 1]$ is also impossible as E^{R}_{half} would yield a suitable q. We conclude that $[x, y] = [2, N]$ or $[1, N - 1]$, and so $\mathcal{R} \setminus S$ must be reduced to a single point.

It remains to consider line 24. At this step of the algorithm, a loop invariant is that the variable $I(i)$ holds the set of records whose value lies in $[1, i]$, *i.e.* $I(i) = \mathcal{S}_{[1,i]}$. At iteration i, $\mathcal{Q}'$ is the set of all queries q overlapping $I(i)$ such that $q \setminus I(i)$ is non-empty, and T is the intersection of all queries in $\mathcal{Q}'$ minus $I(i)$. An invariant of the loop at line 20 is that $T = \mathcal{S}_{[i+1,j]}$ for some j. The algorithm keeps subtracting queries q overlapping T as long as T remains non-empty, and $q \setminus (T \cup I(i)) \neq \varnothing$. The fail condition at line 24 is triggered iff at the end of this process, the value interval $[i + 1, j]$ corresponding to records in T is not limited to a single point. We now show that this is impossible.

Indeed, if $i < N/2$, then $j > N/2$ is impossible as E^{L}_{half} provides a pair of queries covering all values $[1, N/2]$, so that the intersection of queries in $\mathcal{Q}'$ must be contained in E^{L}_{half}; and $N/2 \geq j > i + 1$ is likewise impossible as the query witnessing E'^{L}_{j} would be subtracted from T during the subsequent loop at line 20. It follows that values covered by T are eventually reduced to $\{i + 1\}$. On the other hand if $i \geq N/2$, then E'^{R}_{i+1} provides a query $q' \in \mathcal{Q}'$ corresponding to an interval of values of the form $[h, i + 1]$ for some $h \leq i$, so the initial value of the set T as the intersection of queries in $\mathcal{Q}'$ is already reduced to the single point $\mathcal{S}_{i+1}$. □

It now remains to lower-bound the probability of $\bigcap_{x=1}^{N/2} E'^{L}_{x}$ and $\bigcap_{y=N/2+1}^{N} E'^{R}_{y}$. As was the case in the proof of Proposition 1, these two events are completely symmetric via the reflection that flips $[1, N]$, replacing k by $N + 1 - k$ and switching the roles of left and right coupons. In particular the two aforementioned events must have the same probability, and we shall limit our attention to $\bigcap_{x=1}^{N/2} E'^{L}_{x}$.

First, observe that there are exactly $N - k$ possible ranges $[k, y] \subseteq [1, N]$ with left end point k and length at least 2. It follows that after drawing a single uniformly random query, we have:

$$\Pr\left[E'^{L}_{x}\right] = \frac{2(N - x)}{N(N + 1)}$$

and after Q queries:

$$\Pr\left[\neg E'^{L}_{x}\right] = \left(1 - \frac{2(N - x)}{N(N + 1)}\right)^{Q}. \tag{7}$$

We have:

$$\begin{aligned}
\Pr\left[\bigcap_{x=1}^{N/2} E_x'^L\right] &\geq 1 - \sum_{x=1}^{N/2} \Pr\left[\neg E_x'^L\right] \\
&= 1 - \sum_{x=1}^{N/2} \left(1 - \frac{2(N-x)}{N(N+1)}\right)^Q && \text{by (7)} \\
&\geq 1 - \frac{N}{2}\left(1 - \frac{1}{N+1}\right)^Q \\
&\geq 1 - \frac{N}{2} e^{-Q/(N+1)}.
\end{aligned}$$

Hence we get:

$$\begin{aligned}
\Pr[E] &\geq 1 - 2\Pr\left[\neg E^R_{half}\right] - 2\Pr\left[\neg \bigcap_{x=1}^{N/2} E_x'^L\right] && \text{by Claim 2} \\
&\geq 1 - 4e^{-Q/(2N+2)} - Ne^{-Q/(N+1)}. && \text{by Lemma 2}
\end{aligned}$$

This concludes the proof of the first statement.

Let T be the random variable denoting the first time t such that the event E occurs given uniformly distributed range queries $q_1, \ldots, q_t$. What we have proven so far is $\Pr[T > Q] \leq 4e^{-Q/(2N+2)} + Ne^{-Q/(N+1)}$. It remains to compute $\mathbb{E}[T]$.

$$\begin{aligned}
\mathbb{E}[T] &= \sum_{t=0}^{\infty} \Pr[T > t] \\
&\leq (N+1)\log N + 1 + \sum_{t=\lceil (N+1)\log N \rceil}^{\infty} \left(4e^{-t/(2N+2)} + Ne^{-t/(N+1)}\right). && (8)
\end{aligned}$$

Let us focus on the two sub-terms appearing in the sum:

$$\begin{aligned}
\sum_{t=\lceil (N+1)\log N \rceil}^{\infty} e^{-t/(2N+2)} &\leq \frac{1}{\sqrt{N}} \sum_{t=0}^{\infty} e^{-t/(2N+2)} \\
&= \frac{1/\sqrt{N}}{1 - e^{-1/(2N+2)}} \\
&\leq \frac{1}{\sqrt{N}}(2N+3) \\
&\leq 2\sqrt{N} + 1 && \text{for } N \geq 9 \\
N \sum_{t=\lceil (N+1)\log N \rceil}^{\infty} e^{-t/(N+1)} &\leq \sum_{t=0}^{\infty} e^{-t/(N+1)} \\
&= \frac{1}{1 - e^{-1/(N+1)}} \\
&\leq N + 2.
\end{aligned}$$

where lines 3 and 7 use the fact that $1/(1 - e^{-1/x}) \leq 1 + x$ for $x \geq 0$.

Reinjecting into (8), we conclude that for $N \geq 9$:

$$\mathbb{E}[T] \leq (N+1)\log(N) + N + 8\sqrt{N} + 7.$$

For $N \geq 26$, $8\sqrt{N} + \log(N) + 7 \leq 2N$ so in that case we have:

$$\mathbb{E}[T] \leq N(\log(N) + 3). \qquad \square$$

B.4 Upper Bound for the Approximate Reconstruction Attack

Proposition 3. *Assume N is a multiple of 4, $\epsilon N/2$ is an integer, and $\epsilon < 3/4$. Assume that the database is dense, and range queries are drawn uniformly at random. Then the probability of success of Algorithm 3 after Q queries is lower-bounded by:*

$$1 - 4e^{-Q/(2N+2)} - 2e^{-Q\log(1+\epsilon/2)+\epsilon N/2\cdot(\log(1/\epsilon)+1.5)}.$$

Moreover, the expected number of queries before the algorithm succeeds is upper-bounded by:

$$\frac{5}{4}N\log(1/\epsilon) + O(N).$$

Concretely, for $N \geq 40$, it is upper-bounded by $(1+\epsilon/4)N\log(1/\epsilon) + (2+4\sqrt{\epsilon})N + 4/\epsilon$.

Proof. Let $\mathcal{Q} = \{q_k = (\mathcal{M}_k) : k \leq Q\}$ denote the set of queries. For each $k \leq Q$, let $[x_k, y_k] \subseteq [1, N]$ denote the range of values on which the k-th query q_k bears, *i.e.* $\mathcal{M}_k = \{r \in \mathcal{R} : \mathsf{val}(r) \in [x_k, y_k]\}$.

The proof follows the outline of the proofs of Propositions 1 and 2. Let E denote the event that Algorithm 3 succeeds. We are going to define several events whose joint occurrence implies E. These events will be similar to those used in the proofs of Propositions 1 and 2. Then in a second step we will lower-bound their joint probability.

Define the following events:

- E^L_{half}: there exist two queries q_k, $q_{k'}$ such that $[x_k, y_k] \cup [x_{k'}, y_{k'}] = [1, N/2+1]$ and $[x_k, y_k] \cap [x_{k'}, y_{k'}] \neq \varnothing$.
- E^R_{half}: there exist two queries q_k, $q_{k'}$ such that $[x_k, y_k] \cup [x_{k'}, y_{k'}] = [N/2+1, N]$ and $[x_k, y_k] \cap [x_{k'}, y_{k'}] \neq \varnothing$.
- E''^L_x: there exists a query q_k with $x_k = x \leq N/2$ and $y_k > N/2$.
- E''^R_y: there exists a query q_k with $x_k \leq N/2$ and $y_k = y > N/2$.

As in previous proofs, we call E''^L_x a *left* coupon, and E''^R_y a *right* coupon. The idea is that if we collect $(1-\epsilon)N/2$ left coupons in $[1, N/2]$ and $(1-\epsilon)N/2$ right coupons in $[N/2+1, N]$, in addition to E^L_{half} and E^R_{half}, then the algorithm succeeds. This is expressed by the following claim.

Claim 3. *Let C_L (resp. C_R) denote the event that there exist $(1-\epsilon)N/2$ distinct values $x \leq N/2$ (resp. $y > N/2$) such that E''^L_x (resp. E''^R_y) occurs. Then:*

$$E \supseteq E^L_{half} \cap E^R_{half} \cap C_L \cap C_R.$$

Proof. Throughout, we assume that events E^L_{half}, E^R_{half}, C_L, and C_R occur. Our goal is to show that E happens, *i.e.* Algorithm 3 succeeds. This is rather straightforward, as the algorithm is designed precisely to take advantage of the aforementioned events.

Algorithm 3 can only fail at line 30, if no suitable record r was found during the course of the main loop. Let r now denote any record with value $N/2+1$. We claim that the algorithm must succeed for this particular choice of r, and hence the fail condition at line 30 is never triggered.

We now show that the algorithm succeeds for r such that $\mathsf{val}(r) = N/2+1$. First, $E^L_{half} \cap E^R_{half}$ implies that the variable M at line 2 is reduced to the set $\mathcal{S}_{N/2+1}$ of records with value $N/2+1$. The event $E^L_{half} \cap E^R_{half}$ also implies that the test on line 6 is satisfied. Then C_L implies that $n_L \geq (1-\epsilon)N/2$. Likewise C_R implies $n_R \geq (1-\epsilon)N/2 - 1$; the minus 1 in the last expression is due to the fact the value $N/2+1$ may account for one of the right coupons, whereas sets in $\mathcal{R} \setminus \mathrm{half}_L$ contain records with values in $[N/2+2, N]$ (as M is excluded). Thus $n_L + n_R + 1 \geq (1+\epsilon)N$, so the test at line 15 is also satisfied, which means that the algorithm has succeeded in finding a solution. □

It now remains to study the probability of C_L and C_R. As was the case in the proofs of Propositions 1 and 2, these two events are symmetric and hence must have the same probability, so we focus on C_L.

Let T_k denote the random variable such that the k-th distinct left coupon in $[1, N/2]$ is collected at time T_k, with the convention $T_0 = 0$. Let $\Delta_k = T_{k+1} - T_k$. Notice that C_L happens iff $T_{(1-\epsilon)N/2} \leq Q$. To simplify notation we will write $T_{(1-\epsilon)N/2}$ simply as T.

Suppose k left coupons have been collected for some $k \geq 0$. Then there are $(N/2 - k) \cdot N/2$ ranges in $[1, N]$ that constitute a new left coupon. Hence the probability of drawing a new left coupon is:

$$p_k := \frac{(N/2 - k) \cdot N/2}{N(N+1)/2} = \frac{N/2 - k}{N+1}.$$

The variable Δ_k follows a geometric distribution with probability p_k.

We wish to upper-bound the probability of the event $\neg C_L$, which is equivalent to $T > Q$. For this purpose we note that T is a sum of the independent variables Δ_k, and use the Chernoff bound:

$$\Pr[T > Q] \leq e^{-xQ} \prod_{k=0}^{(1-\epsilon)N/2-1} \mathbb{E}\left[e^{x\Delta_k}\right] \tag{9}$$

for any $x > 0$ (which we will soon choose). Since the Δ_k's are geometrically distributed with probability p_k, their moment-generating function $\mathbb{E}\left[e^{x\Delta_k}\right]$ satisfies:

$$\mathbb{E}\left[e^{x\Delta_k}\right] = \frac{p_k e^x}{1 - (1 - p_k)e^x}$$

with the constraint $e^x < 1/(1 - p_k)$. The most demanding constraint arises for $k = (1-\epsilon)N/2 - 1$, namely:

$$\begin{aligned} e^x &< \frac{1}{1 - p_{(1-\epsilon)N/2-1}} \\ &= \frac{N+1}{(1-\epsilon/2)N}. \end{aligned}$$

We set $x = \log(1 + \epsilon/2)$,[19] which is easily verified to satisfy that constraint; indeed:

$$(1 - \epsilon/2)e^x = 1 - \epsilon^2/4 < (N+1)/N.$$

Reinjecting into the Chernoff bound (9) gives:

$$\Pr[\neg C_L] \leq e^{-(Q-(1-\epsilon)N/2)x} \prod_{k=0}^{(1-\epsilon)N/2-1} \frac{p_k}{1 - (1 - p_k)e^x}. \tag{10}$$

We now focus on the product P appearing on the right-hand side of the previous equation:

$$\begin{aligned} P &:= \prod_{k=0}^{(1-\epsilon)N/2-1} \frac{p_k}{1 - (1 - p_k)(1 + \epsilon/2)} \\ &= \prod_{k=0}^{(1-\epsilon)N/2-1} \left(1 - \frac{\epsilon}{2}\left(\frac{1}{p_k} - 1\right)\right)^{-1} \\ &= \prod_{k=0}^{(1-\epsilon)N/2-1} \left(1 - \frac{\epsilon}{2}\left(\frac{N/2 + k + 1}{N/2 - k}\right)\right)^{-1}. \end{aligned}$$

[19] To shed some light into the choice of x: we want x to be "large" since it dictates the rate of exponential decrease in the bound, but e^x has to be strictly lower than $1/(1 - \epsilon/2) \cdot (N + 1)/N$. The idea is that the term $1/(1 - \epsilon/2)$ is the important one (the other term would essentially disappear if we were to collect even one more coupon), and $e^x = 1 + \epsilon/2$ is a first-order approximation of this upper bound, which conveniently happens to be lower than it since $1/(1 - \epsilon/2) = 1 + \epsilon/2 + \epsilon^2/4 + \ldots$

In this formula, as one would expect, the highest contributing terms occur as k gets close to its upper bound (as collecting the last coupons is more expensive than the first). Before continuing, we perform the change of variables $s := (1-\epsilon)N/2 - k$, which sends the most significant terms to $s = 0, 1, 2, \ldots$ (and, in our view, improves the readability of the formula).

$$\begin{aligned}
P &= \prod_{s=1}^{(1-\epsilon)N/2} \left(1 - \frac{\epsilon}{2}\left(\frac{N - \epsilon N/2 - s + 1}{\epsilon N/2 + s}\right)\right)^{-1} \\
&= \prod_{s=1}^{(1-\epsilon)N/2} \left(\frac{\epsilon}{2} + \frac{s - \epsilon/2}{s + \epsilon N/2}\right)^{-1} \\
\log(P) &= \sum_{s=1}^{(1-\epsilon)N/2} - \log\left(\frac{\epsilon}{2} + \frac{s - \epsilon/2}{s + \epsilon N/2}\right) \\
&\leq \int_{s=0}^{(1-\epsilon)N/2} - \log\left(\frac{\epsilon}{2} + \frac{s - \epsilon/2}{s + \epsilon N/2}\right). \qquad (11)
\end{aligned}$$

Note that the logarithm remains well-defined thanks to the assumption $\epsilon \geq 2/N$. It is straightforward to check that a primitive integral of the above function may be written as $F_1(s) + F_2(s) + F_3(s)$ with:

$$\begin{aligned}
F_1(s) &= \qquad -s \log\left(\frac{\epsilon}{2} + \frac{s - \epsilon/2}{s + \epsilon N/2}\right) \\
F_2(s) &= \qquad \frac{\epsilon N}{2} \log\left(s + \frac{\epsilon N}{2}\right) \\
F_3(s) &= -\frac{\epsilon/2(N\epsilon/2 - 1)}{1 + \epsilon/2} \log\left(s + \frac{\epsilon/2(N\epsilon/2 - 1)}{1 + \epsilon/2}\right).
\end{aligned}$$

Let us start with the contribution of F_1:

$$\begin{aligned}
F_1((1-\epsilon)N/2) - F_1(0) &= -\frac{(1-\epsilon)N}{2} \log\left(1 - \epsilon\left(\frac{1}{2} + \frac{1}{N}\right)\right) \\
&\leq \frac{\epsilon N}{2}(1-\epsilon)\left(1 + \frac{2}{N}\right) \\
&\leq \frac{\epsilon N}{2}\left(1 + \frac{2}{N} - \epsilon - \frac{N\epsilon}{2}\right) \\
&\leq \frac{\epsilon N}{2} \qquad (12)
\end{aligned}$$

where the second line uses $-\log(1-t) < 2t$ for $t < 3/4$ (recall $\epsilon < 3/4$ is an assumption); and the last line uses $\epsilon \geq 2/N$. We turn to F_2 and F_3:

$$\begin{aligned}
F_2((1-\epsilon)N/2) - F_2(0) &= \frac{\epsilon N}{2} \log\left(\frac{1}{\epsilon}\right) \qquad (13) \\
F_3((1-\epsilon)N/2) - F_3(0) &= -\frac{\epsilon/2(N\epsilon/2 - 1)}{1 + \epsilon/2} \log\left(1 + (1-\epsilon)\left(1 + \frac{\epsilon}{2}\right)\frac{N}{2}\right).
\end{aligned}$$

The contribution of F_3 is clearly negative; since we are trying to upper-bound P, we discard it.

Reinjecting (12) and (13) into (11) leaves us with:

$$\log(P) \leq \frac{\epsilon N}{2}\left(\log\left(\frac{1}{\epsilon}\right) + 1\right).$$

Finally, reinjecting this last inequality into (10) gives:

$$\log\left(\Pr\left[\neg C_L\right]\right) \leq -Q \log\left(1 + \frac{\epsilon}{2}\right) + \frac{(1-\epsilon)N}{2} \log\left(1 + \frac{\epsilon}{2}\right) + \frac{\epsilon N}{2}\left(\log\left(\frac{1}{\epsilon}\right) + 1\right).$$

Using $\log(1+x) \leq x$ we can upper-bound the middle summand by $\epsilon N/4$, which simplifies the bound into:

$$\log\left(\Pr\left[\neg C_L\right]\right) \leq -Q \log\left(1 + \frac{\epsilon}{2}\right) + \frac{\epsilon N}{2}\left(\log\left(\frac{1}{\epsilon}\right) + 1.5\right).$$

Combining this last inequality with Lemma 2 yields the desired result:

$$\begin{aligned}\Pr\left[E\right] &\geq 1 - 2\Pr\left[\neg E^R_{half}\right] - 2\Pr\left[\neg C_L\right] \\ &\geq 1 - 4e^{-Q/(2N+2)} - 2e^{-Q\log(1+\epsilon/2)+\epsilon N/2\cdot(\log(1/\epsilon)+1.5)}.\end{aligned}$$

This concludes the proof of the first statement.

Let T' be the random variable denoting the first time t such that the event E occurs given uniformly distributed range queries $q_1, \ldots, q_t$. What we have proven so far is:

$$\Pr\left[T' > Q\right] \leq 4e^{-Q/(2N+2)} + 2e^{-Q\log(1+\epsilon/2)+\epsilon N/2\cdot(\log(1/\epsilon)+1.5)}. \tag{14}$$

It remains to compute $\mathbb{E}\left[T'\right]$. Let us define:

$$s := \frac{\epsilon/2}{\log(1+\epsilon/2)} N(\log(1/\epsilon) + 1.5).$$

Thus, s is chosen so that setting $Q = s$ cancels the exponent in the second (and dominant) term in (14). We have:

$$\begin{aligned}\mathbb{E}\left[T'\right] &= \sum_{t=0}^{\infty} \Pr\left[T' > t\right] \\ &\leq s + 1 + \sum_{t=\lceil s \rceil}^{\infty} \left(4e^{-Q/(2N+2)} + 2e^{-Q\log(1+\epsilon/2)+\epsilon N/2\cdot(\log(1/\epsilon)+1.5)}\right).\end{aligned} \tag{15}$$

Let us focus on the two sub-terms appearing in the sum:

$$\begin{aligned}\sum_{t=\lceil s \rceil}^{\infty} e^{-t/(2N+2)} &\leq e^{-\frac{N}{2N+2}(\log(1/\epsilon)+1.5)} \sum_{t=0}^{\infty} e^{-t/(2N+2)} \\ &= \frac{\sqrt{\epsilon} \cdot e^{-\frac{3}{4}+(\log(1/\epsilon)/2+\frac{3}{4})/(N+1)}}{1 - e^{-1/(2N+2)}} \\ &\leq \frac{1}{2}\sqrt{\epsilon}(2N+3) && \text{for } N \geq 39 \\ \sum_{t=\lceil s \rceil}^{\infty} e^{-t\log(1+\epsilon/2)+\epsilon N/2\cdot(\log(1/\epsilon)+1.5)} &\leq \sum_{t=0}^{\infty} e^{-t\log(1+\epsilon/2)} \\ &= \frac{1}{1-(1+\epsilon/2)^{-1}} \\ &= \frac{2}{\epsilon} + 1.\end{aligned}$$

where the first line uses $x/\log(1+x) \geq 1$, and line 3 uses $1/(1-e^{-1/x}) \leq 1 + x$ for $x \geq 0$, as well as $\epsilon \geq 2/N$.

Reinjecting into (15), we conclude that for $N \geq 39$:

$$\mathbb{E}\left[T'\right] \leq \frac{\epsilon/2}{\log(1+\epsilon/2)} N(\log(1/\epsilon) + 1.5) + \sqrt{\epsilon}(4N+6) + \frac{4}{\epsilon} + 3.$$

Using the fact $x/(\log(1+x)) \leq 1 + x/2$ for $x \geq 0$, and $\epsilon < 1$, we get the desired result:

$$\begin{aligned}\mathbb{E}\left[T'\right] &\leq (1+\epsilon/4)N(\log(1/\epsilon) + 1.5) + 4\sqrt{\epsilon}N + 4/\epsilon + 9 \\ &\leq (1+\epsilon/4)N\log(1/\epsilon) + (2 + 4\sqrt{\epsilon})N + 4/\epsilon\end{aligned}$$

where the last line uses $N \geq 40$ and $\epsilon < 3/4$. □

C Lower Bounds on Data Complexity

Let $\mathcal{Q} = (\mathcal{M}_k)$ denote a set of queries. Rank information will play no role in this section, and our results hold with or without it. Recall from Section 2.3 that the *partition of records* $\mathcal{P} = \{P(r) : r \in \mathcal{R}\}$ is defined by:

$$P(r) = \left(\bigcap_{\{k:r\in\mathcal{M}_k\}} \mathcal{M}_k \right) \setminus \left(\bigcup_{\{k:r\notin\mathcal{M}_k\}} \mathcal{M}_k \right).$$

Likewise, the *partition of values* $\mathcal{V} = \{V(r) : r \in \mathcal{R}\}$ is defined by:

$$V(r) = \left(\bigcap_{\{k:r\in\mathcal{M}_k\}} [x_k, y_k] \right) \setminus \left(\bigcup_{\{r\notin\mathcal{M}_k\}} [x_k, y_k] \right).$$

Also recall that there is a one-to-one correspondence between $\mathcal{P}$ and $\mathcal{V}$ (induced by val). In particular $|\mathcal{P}| = |\mathcal{V}|$.

The number $k = |\mathcal{P}| \leq N$ plays a crucial role: any algorithm operating on $\mathcal{Q}$ can only meaningfully separate k distinct classes among records. Intuitively, it is clear that such an algorithm can only assign values to records up to a precision of $N - k$. In this section, we lower-bound the expected number of queries required to reach a given threshold $|\mathcal{P}| \geq k$. We then formalise the previous intuition to deduce a lower bound on the expected number of queries necessary to assign values to records with precision of $N - k$.

Proposition 4. *Let $\mathcal{P}$ denote the partition of records induced by a set of uniformly random range queries. Let $k \leq N$. Then the expected number of queries necessary to attain $|\mathcal{P}| \geq k$ is lower-bounded by:*

$$\frac{1}{2} N \log\left(\frac{N+1}{N+2-k}\right) - \frac{1}{2}.$$

Proof. As already noted, the partition of values is in bijection with the partition of records. In this proof we focus on the partition of values. When a range query is issued on some range $[x, y] \subseteq [1, N]$, this query potentially allows to separate values below or above x, and below or above y. Following this observation, let us define $S = [0, N]$ as the set of *separators*. We say that a range $[x, y]$ yields the separators $x - 1$ and y. The idea is that a separator $s \in S$ allows to separate values $\leq s$ and $> s$.

The set of all records is assumed to be known. Since we are looking to lower-bound the required number of queries necessary for an attack, this is a fair assumption (we are assuming more information is available). As a result, the separators 0 and N bring no information, and we assume they are known from the start. Define *inner* separators as separators in $[1, N - 1]$. The crucial point is that if we have collected k distinct inner separators, then we are able to partition the values $[1, N]$ into at most $k + 1$ sub-intervals. Indeed, we start knowing 1 interval of values, namely $[1, N]$; and each new, distinct inner separator splits the interval in which it falls in two.

Hence, in order to lower-bound the expected number of queries required to reach $|\mathcal{P}| \geq k$, it suffices to lower-bound the number of queries necessary to collect $k-1$ distinct inner separators. This is another variant of the coupon collector's problem, viewing inner separators as coupons. In the remainder we will use the terms inner separator and coupon interchangeably. Our goal is to lower-bound the number of trials necessary to collect $k - 1$ coupons (among $N - 1$ possible coupons). Each trial consists in drawing a uniform range $[x, y] \subseteq [1, N]$ and collecting the coupon(s) $\{x - 1, y\} \setminus \{0, N\}$.

The distribution of each trial can be characterised as follows. Note that there are $M = N(N+1)/2$ choices of $[x, y]$. Let $S \subseteq [1, N - 1]$ be a set of coupons of size at most two. Then the probability that we collect exactly S during a given trial is:

- $p_0 = 1/M$ if $|S| = 0$: indeed this occurs iff $[x, y] = [1, N]$;
- $p_1 = 2/M$ if $|S| = 1$: indeed letting $S = \{s\}$, this occurs iff $[x, y] = [1, s]$ or $[x, y] = [s + 1, N]$;
- $p_2 = 1/M$ if $|S| = 2$: indeed letting $S = \{s, t\}$, this occurs iff $[x, y] = [s + 1, t]$.

Let us denote by D the above distribution (on subsets of coupons of size at most two).

On the other hand, let D' denote the distribution induced by drawing two coupons independently and uniformly at random, and taking their union. Note that there are $M' = (N-1)^2$ possible pairs of coupons. For $S \subseteq [1, N-1]$ of size at most 2, the probability of drawing S for this new distribution is defined by:

- $p'_0 = 0$ if $|S| = 0$;
- $p'_1 = 1/M'$ if $|S| = 1$: indeed letting $S = \{s\}$, this occurs iff we draw the pair (s, s);
- $p'_2 = 2/M'$ if $|S| = 2$: indeed letting $S = \{s, t\}$, this occurs iff we draw the pair (s, t) or (t, s).

The point of D' is that it amounts to a sequence of two trials in the standard coupon collector's problem with a uniform distribution. On the other hand, the following claim shows that, for our purpose, we can effectively replace D by the much more agreeable D'.

Claim 4. *Let T_k^D (resp. $T_k^{D'}$) denote the number of trials until k coupons have been collected with distribution D (resp. D'). Then $\mathbb{E}\left[T_k^D\right] \geq \mathbb{E}\left[T_k^{D'}\right]$.*

Proof. Let $\mathcal{C}$ denote the set of subsets of coupons of size at most two (so that D and D' are distributions on $\mathcal{C}$). Define the probabilistic mapping $f : \mathcal{C} \to \mathcal{C}$ by:

$$f(S) = S' \text{ where } \begin{cases} \text{if } S = \varnothing, \text{ draw } S' \leftarrow D'; \\ \text{if } S = \{s\} \begin{cases} \text{with probability } \frac{N+2}{4(N-1)} \text{ set } S' = S; \\ \text{otherwise } S' = \{s, t\} \text{ with } t \overset{\$}{\leftarrow} [1, N-1] \setminus \{s\}; \end{cases} \\ \text{if } S = \{s, t\}, \text{ set } S' = S. \end{cases}$$

We claim that f sends the distribution D to D'. To see this, it suffices to check that D' and $f(D)$ match on subsets of size 0 and 1. Indeed, both are probability distributions, and are uniform among subsets of a given size. Both distributions assign probability 0 to the empty set. As for a given singleton $\{s\}$, D' assigns probability $1/(N-1)^2$ and $f(D)$ assigns:

$$\begin{aligned} p_0 p'_1 + p_1 \frac{N+2}{4(N-1)} &= \frac{2}{N(N+1)} \cdot \frac{1}{(N-1)^2} + \frac{4}{N(N+1)} \cdot \frac{N+2}{4(N-1)} \\ &= \frac{1}{(N-1)^2}. \end{aligned}$$

Moreover f is designed in such a way that it can only *add* new coupons. It follows that the expected number of trials to collect k coupons is lower with distribution $f(D) = D'$ than with distribution D. □

Claim 5.

$$\mathbb{E}\left[T_k^{D'}\right] \geq \frac{N}{2} \log\left(\frac{N+1}{N+1-k}\right) - \frac{1}{2}.$$

Proof. As noted earlier, a single trial with distribution D' is equivalent to two trials in a standard coupon collector's problem with uniform distribution. Let U_k be the number of trials after which k distinct coupons have been collected in a standard coupon collector's problem with $N-1$ elements.

From this perspective:

$$\begin{aligned}
\mathbb{E}\left[T_k^{D'}\right] &= \sum_{t=0}^{\infty} t \Pr\left[(U_k = 2t) \cup (U_k = 2t+1)\right] \\
&= \frac{1}{2}\sum_{t=0}^{\infty} 2t\left(\Pr\left[U_k = 2t\right] + \Pr\left[U_k = 2t+1\right]\right) \\
&= \frac{1}{2}\sum_{t=0}^{\infty} \big(2t \Pr\left[U_k = 2t\right] + (2t+1)\Pr\left[U_k = 2t+1\right]\big) \\
&\quad - \frac{1}{2}\sum_{t=0}^{\infty} \Pr\left[U_k = 2t+1\right] \\
&\geq \frac{1}{2}\sum_{t=0}^{\infty} t \Pr\left[U_k = t\right] - \frac{1}{2}\sum_{t=0}^{\infty} \Pr\left[U_k = t\right] \\
&= \frac{1}{2}(\mathbb{E}\left[U_k\right] - 1). \qquad (16)
\end{aligned}$$

On the other hand, by standard arguments:

$$\begin{aligned}
\mathbb{E}\left[U_k\right] &= N(H_N - H_{N-k}) \\
&= N \sum_{i=N-k+1}^{N} \frac{1}{i} \\
&\geq N \int_{x=N-k+1}^{N+1} \frac{1}{x} \\
&= N(\log(N+1) - \log(N+1-k)).
\end{aligned}$$

Combining this with (16) yields the result. □

Summing up, we have shown that $|\mathcal{P}| \geq k$ implies we have collected at least $k-1$ inner separators, so the expected number of queries to reach $|\mathcal{P}| \geq k$ is lower-bounded by $\mathbb{E}\left[T_{k-1}^{D}\right]$. By Claim 4 this expectation is itself lower-bounded by $\mathbb{E}\left[T_{k-1}^{D'}\right]$, and Claim 5 yields the result. □

Proposition 5. *Let $\mathcal{A}$ denote* any *(computationally unbounded) adversary observing the leakage of a sequence of uniformly random range queries (including rank information). In addition to query leakage, $\mathcal{A}$ takes as input a quantity $0 < \epsilon < 1$. $\mathcal{A}$ outputs either $\perp$ (fail), or correctly assigns to all records r an interval of values containing $\mathsf{val}(r)$ of length at most ϵN (success). Then the expected number of queries that $\mathcal{A}$ must observe before it succeeds is lower-bounded by:*

$$\frac{1}{2}N\log\left(\frac{N+1}{\epsilon N+2}\right) - \frac{1}{2} \quad = \quad \frac{1}{2}N\log\left(\frac{1}{\epsilon}\right) - O(N).$$

Proof. By Proposition 4, it suffices to show that if the partition of records $\mathcal{P}$ has cardinality less than $(1-\epsilon)N$, then it is impossible to assign values to records with precision ϵN. To see this, define:

$$\begin{aligned}
m : \mathcal{P} &\to [1, N] \\
p &\mapsto \min(\{\mathsf{val}(r) : r \in p\}).
\end{aligned}$$

Note that m is injective and order $\mathcal{P}$ by increasing value of m as $p_1, \dots, p_P$, where $P = |\mathcal{P}|$. Define the two value assignments $v, v' : \mathcal{P} \to \mathcal{X}$ characterised respectively by $v(p_i) = \{i\}$, and $v'(p_i) = \{N - P + i\}$. The crucial point is that, given the same range queries, v and v' both leave the leakage observed by the adversary unchanged. Hence the adversary cannot distinguish between these two assignments (or the real assignment val, for that matter), and so it cannot determine (with probability 1) a range of values for a given record with length strictly less than $N - P$.

Thus, in order for the adversary to be able to assign a range of length at most ϵN to all records, it must hold that $P \geq (1-\epsilon)N$. Whence Proposition 4 yields a lower bound on the expected number of queries:

$$\begin{aligned}\frac{1}{2}N\log\left(\frac{N+1}{\epsilon N+2}\right) - \frac{1}{2} &= \frac{1}{2}\left(N\log\left(\frac{1}{\epsilon}\right) + N\log\left(\frac{\epsilon N+\epsilon}{\epsilon N+2}\right) - 1\right)\\ &\geq \frac{1}{2}\left(N\log\left(\frac{1}{\epsilon}\right) - N\log(3) - 1\right)\\ &= \frac{1}{2}N\log\left(\frac{1}{\epsilon}\right) - O(N).\end{aligned}$$

where the inequality in the second line uses $\epsilon N \geq 1$. □

Corollary 1. *If the adversary $\mathcal{A}$ of Proposition 5 wishes to correctly determine the value of all records, then the expected number of queries it must observe is lower-bounded by:*

$$\frac{1}{2}N\log\left(\frac{N+1}{3}\right) - \frac{1}{2} \quad = \quad \frac{1}{2}N\log(N) - O(N).$$

Proof. Set $\epsilon = 1/N$ in Proposition 5. □

Remark 1. Obviously the lower bounds of Proposition 5 and Corollary 1 also hold for adversaries that do not have access to rank information, since in that case the adversary has strictly less information.

Remark 2. The results of Proposition 5 and Corollary 1 do not require that the database be dense, and known to be dense by the adversary. However the same results can be easily adapted to the case where the database is known to be dense *e.g.* by adding the assumption that at least two distinct values are taken by at least N records. This gives us enough leeway to "shift" the set of possible values between these two sets as in the proof of Proposition 5, while maintaining density.

D Data Optimality

We consider algorithms observing the leakage of a sequence of queries, with some specific stated objective; and we restrict our attention to algorithms that are *correct* in the following sense: on any given input, either the algorithm fails, or its output correctly fulfills the stated objective with probability 1. In this context, we say that an algorithm is *data-optimal* iff whenever the algorithm fails on a given input, then it was impossible to fulfill the stated objective given that input from an information theoretic perspective: *i.e.* no (computationally unbounded) correct algorithm given the same input would succeed in reaching the stated objective. Informally, a data-optimal algorithm consumes the minimal amount of data required to reach its stated objective with probability 1, in essentially the strongest possible sense.

In this section, we prove that both our algorithms for full reconstruction,[20] namely Algorithms 1 and 2, are data-optimal in this strong sense. It is worth pointing out that computationally unbounded data-optimal algorithms trivially exist, since an unbounded algorithm could simply try all possible assignments of values to records, and check whether they are consistent with the query leakage. Of course our algorithms terminate in polynomial time.

In the context of the following propositions, we consider an algorithm that receives as input some leakage data induced by range queries $[x_k, y_k]$ on a database defined by a set of records $\mathcal{R}$, where each

[20] We believe a polynomial-time data-optimal algorithm for approximate reconstruction from access pattern also exists, but it is far more complex than Algorithm 3; and by Proposition 5, it would ultimately still require $\Omega(N\log(1/\epsilon))$ queries to succeed.

record $r \in \mathcal{R}$ has value $\mathsf{val}(r) \in \mathcal{X} = [1, N]$. In the remainder we refer to val as the *real* assignment of values. The stated objective is to achieve full reconstruction, *i.e.* recover val. To argue that an algorithm meeting this objective is data-optimal, it is enough to show that if the algorithm fails, then there exists another mapping $\mathsf{val}' \neq \mathsf{val}$ assigning values to records such that the induced leakage would be unchanged. Indeed in that case every correct algorithm must fail given the same query leakage, as it must output the same answer whether the real value assignment is val or val'. Whenever this situation occurs, we say that full reconstruction is impossible.

Proposition 6. *Algorithm 1 is data-optimal.*

Proof. The fact that Algorithm 1 is data-optimal is quite natural. Indeed, the algorithm fails iff the partition $\mathcal{P}$ of records (as defined in Section 2.3) has cardinality less than N. Recall that the partition $\mathcal{P}$ is the set of equivalence classes for the equivalence relation: two records are equivalent iff for all queries, either both records are matched by the query, or neither. Intuitively records belonging to the same element of the partition are indistinguishable given the set of queries.

We now assume the algorithm fails, *i.e.* $|\mathcal{P}| < N$. Assume furthermore that the database is dense. By the pigeonhole principle, at least one element of the partition contains records with at least two distinct values x and x'. Observe that altering the real value assignment val to swap the assignment of x and x' to records preserves the same leakage. Hence full reconstruction is impossible.

If the database is not dense on the other hand, then by definition there exists a value x such that no record has value x. Let x' denote a value such that there exists a record with value x', minimising $|x' - x|$ among possible choices. Then the same reasoning applies, *i.e.* composing val with the transposition that exchanges x and x' leaves the leakage invariant. We conclude again that full reconstruction is impossible. □

In the following proposition, we require $N \geq 3$. This is to exclude the very specific case where there are only two records and two values, and the database is known to be dense. Indeed, in that case any (of the two possible) assignments of two distinct values to the two records is correct up to reflection, so it satisfies the output condition of Algorithm 2. We did not think it meaningful to add a check to Algorithm 2 to handle this specific case, since it is vacuous.

Proposition 7. *For $N \geq 3$, Algorithm 2 is data-optimal.*

Proof. Algorithm 2 can fail at lines 7, 14, and 24. In all cases, we must prove that if the fail condition is triggered, then full reconstruction is impossible.

The fail condition at line 7 is equivalent to the fail condition at line 7 of Algorithm 1. The same reasoning as in Proposition 6 shows that full reconstruction is impossible if this fail condition is triggered (this is where we use the fact that the case $N = R = 2$ is excluded). We now move on to lines 14 and 24.

At this step we have split the set of records into the partition of records $\mathcal{P}$, and $|\mathcal{P}| = N$. It follows that each element of $\mathcal{P}$ contains the set of records matching one specific value. We call the elements of $\mathcal{P}$ *points*. The goal of the algorithm is to sort points according to their value, so that it can assign a value to each point. Recall that this is only possible up to reflection: swapping values i and $N + 1 - i$ leaves the leakage invariant.

In the remainder, it will be more convenient to think of the algorithm as sorting points, rather than assigning values to them. This is of course equivalent since to a given order corresponds only one possible value assignment and conversely (recall there are N points). In particular, in order to argue that full reconstruction is impossible (whenever some fail condition is triggered), we will modify the real ordering of points so that the leakage observed by the algorithm is unaffected. If the new order is not a reflection of the real order, this implies that full reconstruction is impossible.

We now turn to the fail statement on line 14 of Algorithm 2. Assume that the fail condition is triggered. Then the variable S holds a non-empty set of records with the following properties (1) $S' = \mathcal{R} \setminus S$ contains at least two points; and (2) there does not exist a query q such that q overlaps S and splits S' (*i.e.* q overlaps S' but does not contain it). Then we alter the ordering of points in the following way: we reverse the order of points within S'. This new order cannot be a reflection of the original order, since S' contains at least two points, and their relative ordering is changed; while the relative ordering of points between S and S' is unaffected.

On the other hand the new order yields the same leakage as the real order: indeed, the only type of query that could distinguish between these two orders is precisely a query that would split S' while overlapping S. To see this, observe that all other types of queries are either supersets or subsets of S'. The first type contains no information on the ordering of points within S'. The second type cannot break the reflection symmetry of elements within S', for the same reason that we can only solve our overall ordering problem up to reflection. Thus full reconstruction is impossible.

The reasoning for the fail condition on line 24 of Algorithm 2 is very similar. If the fail condition is triggered, we have a set T containing at least two points, such that no query splits T and overlaps $T' = \mathcal{R} \setminus T$. Thus the same reasoning applies. To see that there cannot exist a query q splitting T and overlapping T', first observe that by construction the values of records in $I(i)$ and $I(i) \cup T$ both form an initial segment of the set of values (up to reflection). It follows that any query splitting T can overlap $I(i)$, or $\mathcal{R} \setminus (I(i) \cup T)$, but not both. However the first case is impossible due to line 19, and the second case is ruled out by the loop on line 20. □